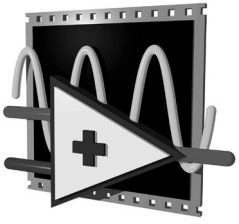


LabVIEW™

# Data Acquisition Basics Manual

NATIONAL INSTRUMENTS™

July 1999 Edition
Part Number 320997D-01

**Worldwide Technical Support and Product Information**

www.natinst.com

**National Instruments Corporate Headquarters**

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 794 0100

**Worldwide Offices**

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, Norway 32 27 73 00, Singapore 2265886,
Spain (Barcelona) 93 582 0251, Spain (Madrid) 91 640 0085, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2377 1200, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to techpubs@natinst.com.

© Copyright 1999 National Instruments Corporation. All rights reserved.

# Important Information

## Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

## Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

## Trademarks

DAQ-STC™, DAQCard™, DAQPad™, LabVIEW™, MITE™, `natinst.com`™, National Instruments™, NI-DAQ™, NI-PGIA™, PXI™, RTSI™, and SCXI™ are trademarks of National Instruments Corporation.

Product and company names mentioned herein are trademarks or trade names of their respective companies.

## WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing for a level of reliability suitable for use in or in connection with surgical implants or as critical components in any life support systems whose failure to perform can reasonably be expected to cause significant injury to a human. Applications of National Instruments products involving medical or clinical treatment can create a potential for death or bodily injury caused by product failure, or by errors on the part of the user or application designer. Because each end-user system is customized and differs from National Instruments testing platforms and because a user or application designer may use National Instruments products in combination with other products in a manner not evaluated or contemplated by National Instruments, the user or application designer is ultimately responsible for verifying and validating the suitability of National Instruments products whenever National Instruments products are incorporated in a system or application, including, without limitation, the appropriate design, process and safety level of such system or application.

# Contents


## About This Manual

Conventions Used in This Manual ..... xxi
LabVIEW Data Types ..... xxiii
Related Documentation ..... xxiv

## PART I
## Before You Get Started

## Chapter 1
## How To Use This Book

## Chapter 2
## Installing and Configuring Your Data Acquisition Hardware

LabVIEW Data Acquisition Hardware Support ..... 2-4
Installing and Configuring Your National Instruments Device ..... 2-14
Installing and Configuring Your DAQ Device Using NI-DAQ 5.*x*, 6.*x* ..... 2-14
Configuring Your DAQ Device Using NI-DAQ 4.8.*x* on the Macintosh ..... 2-14
Installing and Configuring Your SCXI Chassis ..... 2-16
Hardware Configuration ..... 2-16
NI-DAQ 5.*x*, 6.*x* Software Configuration ..... 2-18
NI-DAQ 4.8.*x* for Macintosh Software Configuration ..... 2-18
Configuring Your Channels in NI-DAQ 5.*x*, 6.*x* ..... 2-21

## Chapter 3
## Basic LabVIEW Data Acquisition Concepts

Location of Common DAQ Examples ..... 3-1
Locating the Data Acquisition VIs in LabVIEW ..... 3-3
DAQ VI Organization ..... 3-4
Easy VIs ..... 3-4
Intermediate VIs ..... 3-5
Utility VIs ..... 3-5
Advanced VIs ..... 3-5
VI Parameter Conventions ..... 3-5
Default and Current Value Conventions ..... 3-6
Common DAQ VI Parameters ..... 3-7
Error Handling ..... 3-8

Channel, Port, and Counter Addressing ..... 3-8
Channel Name Addressing ..... 3-9
Channel Number Addressing ..... 3-10
Limit Settings ..... 3-11
Data Organization for Analog Applications ..... 3-14

## Chapter 4
## Where You Should Go Next

Questions You Should Answer ..... 4-3

# Part II
# Catching the Wave with Analog Input

## Chapter 5
## Things You Should Know about Analog Input

Defining Your Signal ..... 5-1
What Is Your Signal Referenced To? ..... 5-2
Grounded Signal Sources ..... 5-2
Floating Signal Sources ..... 5-3
Choosing Your Measurement System ..... 5-4
Resolution ..... 5-4
Device Range ..... 5-5
Signal Limit Settings ..... 5-6
Considerations for Selecting Analog Input Settings ..... 5-7
Differential Measurement System ..... 5-9
Referenced Single-Ended Measurement System ..... 5-11
Nonreferenced Single-Ended Measurement System ..... 5-12
Channel Addressing with the AMUX-64T ..... 5-13
The AMUX-64T Scanning Order ..... 5-14
Important Terms You Should Know ..... 5-17

## Chapter 6
## One-Stop Single-Point Acquisition

Single-Channel, Single-Point Analog Input ..... 6-1
Multiple-Channel Single-Point Analog Input ..... 6-3
Using Analog Input/Output Control Loops ..... 6-6
Using Software-Timed Analog I/O Control Loops ..... 6-6
Using Hardware-Timed Analog I/O Control Loops ..... 6-7
Improving Control Loop Performance ..... 6-9

## Chapter 7
## Buffering Your Way through Waveform Acquisition

Can You Wait for Your Data? ..... 7-1
Acquiring a Single Waveform ..... 7-2
Acquiring Multiple Waveforms ..... 7-3
Simple-Buffered Analog Input Examples ..... 7-5
Simple-Buffered Analog Input with Graphing ..... 7-5
Simple-Buffered Analog Input with Multiple Starts ..... 7-6
Simple-Buffered Analog Input with a Write to Spreadsheet File ..... 7-8
Triggered Analog Input ..... 7-8
Do You Need to Access Your Data during Acquisition? ..... 7-8
Continuously Acquiring Data from Multiple Channels ..... 7-10
Asynchronous Continuous Acquisition Using DAQ Occurrences ..... 7-11
Circular-Buffered Analog Input Examples ..... 7-13
Basic Circular-Buffered Analog Input ..... 7-13
Other Circular-Buffered Analog Input Examples ..... 7-14
Simultaneous Buffered Waveform Acquisition and Waveform Generation ..... 7-14

## Chapter 8
## Controlling Your Acquisition with Triggers

Hardware Triggering ..... 8-1
Digital Triggering ..... 8-2
Digital Triggering Examples ..... 8-4
Digital Triggering Examples ..... 8-5
Analog Triggering ..... 8-6
Analog Triggering Examples ..... 8-8
Software Triggering ..... 8-9
Conditional Retrieval Examples ..... 8-12

## Chapter 9
## Letting an Outside Source Control Your Acquisition Rate

Externally Controlling Your Channel Clock ..... 9-3
Externally Controlling Your Scan Clock ..... 9-6
Externally Controlling the Scan and Channel Clocks ..... 9-9

# PART III
# Making Waves with Analog Output

## Chapter 10
## Things You Should Know about Analog Output

Single-Point Output ..... 10-1
Buffered Analog Output ..... 10-1

## Chapter 11
## One-Stop Single-Point Generation

Single-Immediate Updates ..... 11-1
Multiple-Immediate Updates ..... 11-3

## Chapter 12
## Buffering Your Way through Waveform Generation

Buffered Analog Output ..... 12-1
Changing the Waveform during Generation—Circular-Buffered Output ..... 12-3
Eliminating Errors from Your Circular-Buffered Application ..... 12-5
Buffered Analog Output Examples ..... 12-6

## Chapter 13
## Letting an Outside Source Control Your Update Rate

Externally Controlling Your Update Clock ..... 13-1
Supplying an External Test Clock from Your DAQ Device ..... 13-3

## Chapter 14
## Simultaneous Buffered Waveform Acquisition and Generation

Using E Series MIO Boards ..... 14-1
    Software Triggered ..... 14-1
    Hardware Triggered ..... 14-3
Using Legacy MIO Boards ..... 14-4
    Software Triggered ..... 14-4
    Hardware Triggered ..... 14-5
Using Lab/1200 Boards ..... 14-6

# Part IV
# Getting Square with Digital I/O

## Chapter 15
## Things You Should Know about Digital I/O

Types of Digital Acquisition/Generation ..... 15-2

## Chapter 16
## When You Need It Now—Immediate Digital I/O

## Chapter 17
## Shaking Hands with a Digital Partner

Sending Out Multiple Digital Values ..... 17-3
Nonbuffered Handshaking ..... 17-5
Buffered Handshaking ..... 17-6
Simple Buffered Examples ..... 17-7
Circular-Buffered Examples ..... 17-9

# Part V
# SCXI—Getting Your Signals in Great Condition

## Chapter 18
## Things You Should Know about SCXI

What Is Signal Conditioning? ..... 18-1
Amplification ..... 18-4
Isolation ..... 18-5
Filtering ..... 18-5
Transducer Excitation ..... 18-5
Linearization ..... 18-6

## Chapter 19
## Hardware and Software Setup for Your SCXI System

SCXI Operating Modes ..... 19-5
Multiplexed Mode for Analog Input Modules ..... 19-5
Multiplexed Mode for the SCXI-1200 (Windows) ..... 19-6
Multiplexed Mode for Analog Output Modules ..... 19-6
Multiplexed Mode for Digital and Relay Modules ..... 19-6
Parallel Mode for Analog Input Modules ..... 19-6

Parallel Mode for the SCXI-1200 (Windows) ..... 19-7
Parallel Mode for Digital Modules ..... 19-7
SCXI Software Installation and Configuration ..... 19-8

## Chapter 20
## Special Programming Considerations for SCXI

SCXI Channel Addressing ..... 20-1
SCXI Gains ..... 20-3
SCXI Settling Time ..... 20-5

## Chapter 21
## Common SCXI Applications

Analog Input Applications for Measuring Temperature and Pressure ..... 21-2
Measuring Temperature with Thermocouples ..... 21-2
Temperature Sensors for Cold-Junction Compensation ..... 21-3
Amplifier Offset ..... 21-5
VI Examples ..... 21-6
Measuring Temperature with RTDs ..... 21-10
Measuring Pressure with Strain Gauges ..... 21-13
Analog Output Application Example ..... 21-17
Digital Input Application Example ..... 21-18
Digital Output Application Example ..... 21-19
Multi-Chassis Applications ..... 21-21

## Chapter 22
## SCXI Calibration—Increasing Signal Measurement Precision

EEPROM—Your System's Holding Tank for Calibration Constants ..... 22-1
Calibrating SCXI Modules ..... 22-3
SCXI Calibration Methods for Signal Acquisition ..... 22-4
One-Point Calibration ..... 22-5
Two-Point Calibration ..... 22-6
Calibrating SCXI Modules for Signal Generation ..... 22-7

# PART VI
# Counting Your Way to High-Precision Timing

## Chapter 23
## Things You Should Know about Counters

Knowing the Parts of Your Counter ..... 23-2
Knowing Your Counter Chip ..... 23-4
TIO-ASIC ..... 23-5
DAQ-STC ..... 23-5
Am9513 ..... 23-5
8253/54 ..... 23-5

## Chapter 24
## Generating a Square Pulse or Pulse Trains

Generating a Square Pulse ..... 24-1
TIO-ASIC, DAQ-STC, and Am9513 ..... 24-3
8253/54 ..... 24-4
Generating a Single Square Pulse ..... 24-4
TIO-ASIC, DAQ-STC, Am9513 ..... 24-5
8253/54 ..... 24-7
Generating a Pulse Train ..... 24-10
Generating a Continuous Pulse Train ..... 24-10
TIO-ASIC, DAQ-STC, Am9513 ..... 24-11
8253/54 ..... 24-13
Generating a Finite Pulse Train ..... 24-14
TIO-ASIC, DAQ-STC, Am9513 ..... 24-15
DAQ-STC ..... 24-17
8253/54 ..... 24-18
Counting Operations When All Your Counters Are Used ..... 24-21
Knowing the Accuracy of Your Counters ..... 24-23
8253/54 ..... 24-23
Stopping Counter Generations ..... 24-24
DAQ-STC, Am9513 ..... 24-24
8253/54 ..... 24-24

## Chapter 25
## Measuring Pulse Width

Measuring a Pulse Width ..... 25-1
Determining Pulse Width ..... 25-2
TIO-ASIC, DAQ-STC ..... 25-2
Am9513 ..... 25-4
8253/54 ..... 25-5
Controlling Your Pulse Width Measurement ..... 25-6
TIO-ASIC, DAQ-STC, or Am9513 ..... 25-6
Buffered Pulse and Period Measurement ..... 25-7
Increasing Your Measurable Width Range ..... 25-9

## Chapter 26
## Measuring Frequency and Period

Knowing How and When to Measure Frequency and Period ..... 26-1
TIO-ASIC, DAQ-STC, Am9513 ..... 26-2
8253/54 ..... 26-2
Connecting Counters to Measure Frequency and Period ..... 26-3
TIO-ASIC, DAQ-STC, Am9513 ..... 26-3
Measuring the Frequency and Period of High Frequency Signals ..... 26-4
TIO-ASIC, DAQ-STC ..... 26-4
Am9513 ..... 26-5
TIO-ASIC, DAQ-STC, Am9513 ..... 26-6
8253/54 ..... 26-7
Measuring the Period and Frequency of Low Frequency Signals ..... 26-8
TIO-ASIC, DAQ-STC ..... 26-8
Am9513 ..... 26-9
TIO-ASIC, DAQ-STC, Am9513 ..... 26-10
8253/54 ..... 26-10

## Chapter 27
## Counting Signal Highs and Lows

Connecting Counters to Count Events and Time ..... 27-1
Am9513 ..... 27-2
Counting Events ..... 27-3
TIO-ASIC, DAQ-STC ..... 27-3
Am9513 ..... 27-5
8253/54 ..... 27-7
Counting Elapsed Time ..... 27-8
TIO-ASIC, DAQ-STC ..... 27-8
Am9513 ..... 27-10
8253/54 ..... 27-12

## Chapter 28
## Dividing Frequencies

TIO-ASIC, DAQ-STC, Am9513....28-2
8253/54....28-3

# PART VII
# Debugging Your Data Acquisition Application

## Chapter 29
## Debugging Techniques

Hardware Connection Errors....29-1
Software Configuration Errors....29-1
VI Construction Errors....29-2
Error Handling....29-2
Single-Stepping through a VI....29-3
Execution Highlighting....29-3
Using the Probe Tool....29-4
Setting Breakpoints and Showing Advanced DAQ VIs....29-4

## Appendix A
## LabVIEW Data Acquisition Common Questions

## Appendix B
## Technical Support Resources

## Glossary

## Index

# Figures

Figure 2-1. Installing and Configuring DAQ Devices ..... 2-2
Figure 2-2. How NI-DAQ Relates to Your System and DAQ Devices ..... 2-3
Figure 2-3. NI-DAQ Device Window Listing ..... 2-15
Figure 2-4. Accessing the Device Configuration Window in NI-DAQ ..... 2-15
Figure 2-5. Device Configuration and I/O Connector Windows in NI-DAQ ..... 2-16
Figure 2-6. Accessing the NI-DAQ SCXI Configuration Window ..... 2-18
Figure 2-7. SCXI Configuration Window in NI-DAQ ..... 2-19

Figure 3-1. Accessing the Data Acquisition Palette ..... 3-3
Figure 3-2. Data Acquisition VIs Palette ..... 3-3
Figure 3-3. Analog Input VI Palette Organization ..... 3-4
Figure 3-4. LabVIEW Help Window Conventions ..... 3-6
Figure 3-5. LabVIEW Error In Input and Error Out Output Error Clusters ..... 3-8
Figure 3-6. Channel String Controls ..... 3-9
Figure 3-7. Channel String Array Controls ..... 3-11
Figure 3-8. Limit Settings, Case 1 ..... 3-12
Figure 3-9. Limit Settings, Case 2 ..... 3-13
Figure 3-10. Example of a Basic 2D Array ..... 3-14
Figure 3-11. 2D Array in Row Major Order ..... 3-14
Figure 3-12. 2D Array in Column Major Order ..... 3-15
Figure 3-13. Extracting a Single Channel from a Column Major 2D Array ..... 3-15
Figure 3-14. Analog Output Buffer 2D Array ..... 3-16

Figure 5-1. Types of Analog Signals ..... 5-1
Figure 5-2. Grounded Signal Sources ..... 5-2
Figure 5-3. Floating Signal Sources ..... 5-3
Figure 5-4. The Effects of Resolution on ADC Precision ..... 5-4
Figure 5-5. The Effects of Range on ADC Precision ..... 5-5
Figure 5-6. The Effects of Limit Settings on ADC Precision ..... 5-6
Figure 5-7. 8-Channel Differential Measurement System ..... 5-9
Figure 5-8. Common-Mode Voltage ..... 5-10
Figure 5-9. 16-Channel RSE Measurement System ..... 5-11
Figure 5-10. 16-Channel NRSE Measurement System ..... 5-12

Figure 6-1. AI Sample Channel VI ..... 6-1
Figure 6-2. Acquiring Data Using the Acquire 1 Point from 1 Channel VI ..... 6-2
Figure 6-3. Acquiring a Voltage from Multiple Channels with the AI Sample Channels VI ..... 6-3
Figure 6-4. The AI Single Scan VI Help Diagram ..... 6-4
Figure 6-5. Using the Intermediate VIs for a Basic Non-Buffered Application ..... 6-4
Figure 6-6. The Cont Acq&Chart (Immediate) VI Block Diagram ..... 6-5
Figure 6-7. Software-Timed Analog I/O ..... 6-7
Figure 6-8. Analog IO Control Loop (HW-Timed) VI Block Diagram ..... 6-8

Figure 7-1. How Buffers Work ..... 7-2
Figure 7-2. The AI Acquire Waveform VI ..... 7-3
Figure 7-3. The AI Acquire Waveforms VI ..... 7-3
Figure 7-4. Using the Intermediate VIs to Acquire Multiple Waveforms ..... 7-4
Figure 7-5. Simple Buffered Analog Input Example ..... 7-5
Figure 7-6. Simple Buffered Analog Input with Graphing ..... 7-6
Figure 7-7. Taking a Specified Number of Samples with the Intermediate VIs ..... 7-7
Figure 7-8. Writing to a Spreadsheet File after Acquisition ..... 7-8
Figure 7-9. How a Circular Buffer Works ..... 7-9
Figure 7-10. Continuously Acquiring Data with the Intermediate VIs ..... 7-11
Figure 7-11. Continuous Acq&Chart (Async Occurrence) VI ..... 7-12
Figure 7-12. Basic Circular-Buffered Analog Input Using the Intermediate VIs ..... 7-13

Figure 8-1. Diagram of a Digital Trigger ..... 8-2
Figure 8-2. Digital Triggering with Your DAQ Device ..... 8-3
Figure 8-3. Block Diagram of the Acquire N Scans Digital Trig VI ..... 8-4
Figure 8-4. Block Diagram of the Acquire N Scans Digital VI Using NI 406*x* Hardware ..... 8-5
Figure 8-5. Diagram of an Analog Trigger ..... 8-6
Figure 8-6. Analog Triggering with Your DAQ Device ..... 8-7
Figure 8-7. Block Diagram of the Acquire N Scans Analog Hardware Trig VI ..... 8-8
Figure 8-8. Timeline of Conditional Retrieval ..... 8-10
Figure 8-9. The AI Read VI Conditional Retrieval Cluster ..... 8-11
Figure 8-10. Block Diagram of the Acquire N Scans Analog Software Trig VI ..... 8-12

Figure 9-1. Channel and Scan Intervals Using the Channel Clock ..... 9-1
Figure 9-2. Round-Robin Scanning Using the Channel Clock ..... 9-2
Figure 9-3. Example of a TTL Signal ..... 9-3
Figure 9-4. Getting Started Analog Input Example VI ..... 9-4
Figure 9-5. Setting the Clock Source Code for External Conversion Pulses for E Series Devices ..... 9-5
Figure 9-6. Example Block Diagram of Acquire N Scans Digital External Sample Trigger VI ..... 9-6
Figure 9-7. Externally Controlling Your Scan Clock with the Getting Started Analog Input Example VI ..... 9-8
Figure 9-8. Controlling the Scan and Channel Clock Simultaneously ..... 9-9

Figure 11-1. Single Immediate Update Using the AO Update Channels VI ..... 11-1
Figure 11-2. Single Immediate Update Using the AO Update Channel VI ..... 11-2
Figure 11-3. Single Immediate Update Using Intermediate VI ..... 11-2
Figure 11-4. Multiple Immediate Updates Using Intermediate VI ..... 11-3

Figure 12-1. Waveform Generation Using the AO Generate Waveforms VI ..... 12-1
Figure 12-2. Waveform Generation Using the AO Waveform Gen VI ..... 12-2

Figure 12-3. Waveform Generation Using Intermediate VIs ..... 12-3
Figure 12-4. Circular Buffered Waveform Generation Using the AO Continuous Gen VI ..... 12-4
Figure 12-5. Circular Buffered Waveform Generation Using Intermediate VIs..... 12-5
Figure 12-6. Display and Output Acq'd File (Scaled) VI ..... 12-6

Figure 13-1. Generate N Updates-ExtUpdateClk VI..... 13-2

Figure 14-1. Simultaneous Input/Output Using the Simul AI/AO Buffered (E-series MIO) VI ..... 14-2
Figure 14-2. Simultaneous Input/Output Using the Simul AI/AO Buffered Trigger (E-series MIO) VI ..... 14-3
Figure 14-3. Simultaneous Input/Output Using the Simul AI/AO Buffered (Legacy MIO) VI ..... 14-4
Figure 14-4. Simultaneous Input/Output Using the Simul AI/AO Buffered Trigger (Legacy MIO) VI ..... 14-5

Figure 15-1. Digital Ports and Lines..... 15-1

Figure 16-1. The Easy Digital VIs..... 16-2

Figure 17-1. Connecting Signal Lines for Digital Input..... 17-3
Figure 17-2. Connecting Digital Signal Lines for Digital Output ..... 17-4
Figure 17-3. Nonbuffered Handshaking Using the DIO Single Read/Write VI..... 17-5
Figure 17-4. Nonbuffered Handshaking Using the DIO Single Read/Write VI..... 17-6
Figure 17-5. Buffered Output Using the DIO-32 Devices ..... 17-7
Figure 17-6. Buffered Output Using DAQ Devices (Other Than DIO-32 Series Devices) ..... 17-8
Figure 17-7. Buffered Input Using DIO-32 Devices ..... 17-8
Figure 17-8. Buffered Input Using DAQ Devices (Other than DIO-32 Devices)..... 17-9
Figure 17-9. Digital Handshaking Using a Circular Buffer ..... 17-10

Figure 18-1. Common Types of Transducers/Signals and Signal Conditioning ..... 18-3
Figure 18-2. Amplifying Signals near the Source to Increase Signal-to-Noise Ratio (SNR) ..... 18-4

Figure 19-1. SCXI System..... 19-2
Figure 19-2. Components of an SCXI System ..... 19-3
Figure 19-3. SCXI Chassis ..... 19-4

Figure 21-1. Continuous Transducer Measurement VI ..... 21-6
Figure 21-2. Measuring a Single Module with the Acquire and Average VI..... 21-7
Figure 21-3. Measuring Temperature Sensors Using the Acquire and Average VI ..... 21-8

Figure 21-4. Continuously Acquiring Data Using Intermediate VIs ..... 21-9
Figure 21-5. Measuring Temperature Using Information from the DAQ Channel Wizard ..... 21-12
Figure 21-6. Measuring Temperature Using the Convert RTD Reading VI ..... 21-13
Figure 21-7. Half-Bridge Strain Gauge ..... 21-14
Figure 21-8. Measuring Pressure Using Information from the DAQ Channel Wizard ..... 21-15
Figure 21-9. Convert Strain Gauge Reading VI ..... 21-15
Figure 21-10. SCXI-1124 Update Channels VI ..... 21-17
Figure 21-11. Inputting Digital Signals through an SCXI Chassis Using Easy Digital VIs ..... 21-17
Figure 21-12. Outputting Digital Signals through an SCXI Chassis Using Easy Digital VIs ..... 21-19

Figure 23-1. Counter Gating Modes ..... 23-4
Figure 23-2. Wiring a 7404 Chip to Invert a TTL Signal ..... 23-6

Figure 24-1. Pulse Duty Cycles ..... 24-2
Figure 24-2. Positive and Negative Pulse Polarity ..... 24-3
Figure 24-3. Pulses Created with Positive Polarity and Toggled Output ..... 24-3
Figure 24-4. Phases of a Single Negative Polarity Pulse ..... 24-4
Figure 24-5. Physical Connections for Generating a Square Pulse ..... 24-5
Figure 24-6. Diagram of Delayed Pulse-Easy (DAQ-STC) VI ..... 24-6
Figure 24-7. Diagram of Delayed Pulse-Int (DAQ-STC) VI ..... 24-7
Figure 24-8. External Connections Diagram from the Front Panel of Delayed Pulse (8253) VI ..... 24-7
Figure 24-9. Frame 0 of Delayed Pulse (8253) VI ..... 24-8
Figure 24-10. Frame 1 of Delayed Pulse (8253) VI ..... 24-9
Figure 24-11. Frame 2 of Delayed Pulse (8253) VI ..... 24-10
Figure 24-12. Physical Connections for Generating a Continuous Pulse Train ..... 24-11
Figure 24-13. Diagram of Cont Pulse Train-Easy (DAQ-STC) VI ..... 24-11
Figure 24-14. Diagram of Cont Pulse Train-Int (DAQ-STC) VI ..... 24-12
Figure 24-15. External Connections Diagram from the Front Panel of Cont Pulse Train (8253) VI ..... 24-13
Figure 24-16. Diagram of Cont Pulse Train (8253) VI ..... 24-14
Figure 24-17. Physical Connections for Generating a Finite Pulse Train ..... 24-14
Figure 24-18. Diagram of Finite Pulse Train-Easy (DAQ-STC) VI ..... 24-15
Figure 24-19. Diagram of Finite Pulse Train-Int (DAQ-STC) VI ..... 24-16
Figure 24-20. External Connections Diagram from the Front Panel of Finite Pulse Train Adv (DAQ-STC) VI ..... 24-17
Figure 24-21. Diagram of Finite Pulse Train-Adv (DAQ-STC) VI ..... 24-18
Figure 24-22. External Connections Diagram from the Front Panel of Finite Pulse Train (8253) VI ..... 24-18
Figure 24-23. Frame 0 of Finite Pulse Train (8253) VI ..... 24-19

Figure 24-24. Frame 1 of Finite Pulse Train (8253) VI ..... 24-20
Figure 24-25. Frame 2 of Finite Pulse Train (8253) VI ..... 24-21
Figure 24-26. CTR Control VI Front Panel and Block Diagram ..... 24-22
Figure 24-27. Uncertainty of One Timebase Period..... 24-23
Figure 24-28. Using the Generate Delayed Pulse and Stopping the Counting Operation..... 24-24
Figure 24-29. Stopping a Generated Pulse Train..... 24-24

Figure 25-1. Counting Input Signals to Determine Pulse Width..... 25-1
Figure 25-2. Physical Connections for Determining Pulse Width ..... 25-2
Figure 25-3. Diagram of Measure Pulse Width (DAQ-STC) VI..... 25-2
Figure 25-4. Menu Choices for Type of Measurement for the Measure Pulse Width or Period (DAQ-STC) VI..... 25-3
Figure 25-5. Diagram of Measure Pulse Width (9513) VI..... 25-4
Figure 25-6. Menu Choices for Type of Measurement for the Measure Pulse Width or Period (9513) VI..... 25-4
Figure 25-7. Diagram of Measure Short Pulse Width (8253) VI ..... 25-5
Figure 25-8. Measuring Pulse Width with Intermediate VIs..... 25-7
Figure 25-9. Diagram of Meas Buffered Pulse-Period (DAQ-STC) VI..... 25-7

Figure 26-1. Measuring Square Wave Frequency ..... 26-1
Figure 26-2. Measuring a Square Wave Period..... 26-2
Figure 26-3. External Connections for Frequency Measurement..... 26-3
Figure 26-4. External Connections for Period Measurement ..... 26-3
Figure 26-5. Diagram of Measure Frequency-Easy (DAQ-STC) VI ..... 26-4
Figure 26-6. Diagram of Measure Frequency-Easy (9513) VI..... 26-5
Figure 26-7. Frequency Measurement Example Using Intermediate VIs ..... 26-6
Figure 26-8. Diagram of Measure Frequency > 1 kHz (8253) VI..... 26-7
Figure 26-9. Diagram of Measure Period-Easy (DAQ-STC) VI..... 26-8
Figure 26-10. Diagram of Measure Period-Easy (9513) VI ..... 26-9
Figure 26-11. Measuring Period Using Intermediate Counter VIs..... 26-10

Figure 27-1. External Connections for Counting Events..... 27-1
Figure 27-2. External Connections for Counting Elapsed Time ..... 27-1
Figure 27-3. External Connections to Cascade Counters for Counting Events..... 27-2
Figure 27-4. External Connections to Cascade Counters for Counting Elapsed Time..... 27-3
Figure 27-5. Diagram of Count Events-Easy (DAQ-STC) VI ..... 27-3
Figure 27-6. Diagram of Count Events-Int (DAQ-STC) VI..... 27-4
Figure 27-7. Diagram of Count Events-Easy (9513) VI..... 27-5
Figure 27-8. Diagram of Count Events-Int (9513) VI..... 27-6
Figure 27-9. Diagram of Count Events (8253) VI..... 27-7
Figure 27-10. Diagram of Count Time-Easy (DAQ-STC) VI..... 27-8
Figure 27-11. Diagram of Count Time-Int (DAQ-STC) VI ..... 27-9

Figure 27-12. Diagram of Count Time-Easy (9315) VI ..... 27-10
Figure 27-13. Diagram of Count Time-Int (9513) VI ..... 27-11
Figure 27-14. Diagram of Count Time (8253) VI ..... 27-12

Figure 28-1. Wiring Your Counters for Frequency Division ..... 28-1
Figure 28-2. Programming a Single Divider for Frequency Division ..... 28-2

Figure 29-1. Error Checking Using the General Error Handler VI ..... 29-2
Figure 29-2. Error Checking Using the Simple Error Handler VI ..... 29-3

## Tables

Table 2-1. LabVIEW DAQ Hardware Support with NI-DAQ ..... 2-4

Table 5-1. Measurement Precision for Various Device Ranges and Limit Settings (12-bit A/D Converter) ..... 5-8
Table 5-2. Analog Input Channel Range ..... 5-13
Table 5-3. Scanning Order for Each DAQ Device Input Channel with One or Two AMUX-64Ts ..... 5-15
Table 5-4. Scanning Order for Each DAQ Device Input Channel with Four AMUX-64Ts ..... 5-16

Table 9-1. External Scan Clock Input Pins ..... 9-7

Table 13-1. External Update Clock Input Pins ..... 13-2

Table 18-1. Phenomena and Transducers ..... 18-1

Table 20-1. SCXI-1100 Channel Arrays, Input Limits Arrays, and Gains ..... 20-4

Table 25-1. Internal Counter Timebases and Their Corresponding Maximum Pulse Width, Period, or Time Measurements ..... 25-9

Table 27-1. Adjacent Counters for Counter Chips ..... 27-2

# About This Manual

The *LabVIEW Data Acquisition Basics Manual* includes the information you need to get started with data acquisition and LabVIEW. You should have a basic knowledge of LabVIEW before you try to read this manual. If you have never worked with LabVIEW, please read through the *LabVIEW QuickStart Guide* or the *LabVIEW Online Tutorial* before you begin. This manual shows you how to configure your software, teaches you basic concepts needed to accomplish your task, and refers you to common example VIs in LabVIEW. If you have used LabVIEW for data acquisition before, you can use this book as a troubleshooting guide.

This manual supplements the *LabVIEW User Manual*, and assumes that you are familiar with that material. You also should be familiar with the operation of LabVIEW, your computer, your computer's operating system, and your data acquisition (DAQ) device.

## Conventions Used in This Manual

The following conventions are used in this manual:

| | |
|---|---|
| <> | Angle brackets that contain numbers separated by an ellipsis represent a range of values associated with a bit or signal name—for example, DBIO<3..0>. |
| [ ] | Square brackets enclose optional items—for example, [`response`]. |
| **»** | The **»** symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box. |
| | This icon denotes a note, which alerts you to important information. |
| | This icon denotes a tip, which alerts you to advisory information. |
| **bold** | Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names. |
| *italic* | Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply. |

| | |
|---|---|
| `monospace` | Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts. |
| **`monospace bold`** | Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples. |
| *`monospace italic`* | Italic text in this font denotes text that is a placeholder for a word or value that you must supply. |
| NI-DAQ 4.8.*x* | NI-DAQ 4.8.*x* refers to functions supported only on the Macintosh for NuBus DAQ products. |
| NI-DAQ 5.*x* | NI-DAQ 5.*x* refers to functions supported only on Windows DAQ products. |
| NI-DAQ 6.0 | NI-DAQ 6.0 refers to functions supported only on Windows and PCI-based Macintosh DAQ products. |
| **Platform** | Text in this font denotes a specific platform and indicates that the text following it applies only to that platform. |

# LabVIEW Data Types

Each VI description gives a data type picture for each input and output parameter, as illustrated in the following table:

| Control | Indicator | Data Type |
|---|---|---|
| | | Signed 8-bit integer |
| | | Signed 16-bit integer |
| | | Signed 32-bit integer |
| | | Unsigned 8-bit integer |
| | | Unsigned 16-bit integer |
| | | Unsigned 32-bit integer |
| | | Single-precision floating-point number |
| | | Double-precision floating-point number |
| | | Extended-precision floating-point number |
| | | String |
| | | Boolean |
| | | Array of signed 32-bit integers |
| | | 2D Array of signed 32-bit integers |
| | | Cluster |
| | | File Refnum |

# Related Documentation

The following documents contain information you might find helpful as you read this manual:

- *LabVIEW QuickStart Guide*
- *LabVIEW User Manual*
- *G Programming Reference Manual*
- *LabVIEW Function and VI Reference Manual*
- LabVIEW *Online Reference*, available online by selecting **Help»Online Reference**
- LabVIEW *Online Tutorial*, which you launch from the LabVIEW dialog box **(Windows)**
- *Application Note 025, Field Wiring and Noise Considerations for Analog Signals*
- *Application Note 046, Measuring Temperature with RTDs—A Tutorial*
- The user manuals for the data acquisition boards you use

# Part I

# Before You Get Started

This section contains all the information you should know before you start learning about data acquisition with LabVIEW.

Part I, *Before You Get Started*, contains the following chapters:

- Chapter 1, *How To Use This Book*, explains how this manual is organized.
- Chapter 2, *Installing and Configuring Your Data Acquisition Hardware*, explains how to set up your system to use data acquisition with LabVIEW and your data acquisition hardware.
- Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, explains key concepts in understanding how data acquisition works with LabVIEW.
- Chapter 4, *Where You Should Go Next*, directs you to the chapter or chapters in the manual best suited to answer questions about your data acquisition application.

# 1

# How To Use This Book

This chapter explains how this manual is organized. The following outline shows you what information you can find in this manual.

**Part I Before You Get Started**

How to Use This Book

Installing and Configuring Your Data Acquisition Hardware

Basic LabVIEW Data Acquisition Concepts

Where You Should Go Next

**Part II Catching the Wave with Analog Input**

Things You Should Know about Analog Input

One-Stop Single-Point Acquisition

Buffering Your Way through Waveform Acquisition

Controlling Your Acquisition with Triggers

Letting an Outside Source Control Your Acquisition Rate

**Part III Making Waves with Analog Output**

Things You Should Know about Analog Output

One-Stop Single-Point Generation

Buffering Your Way through Waveform Generation

Letting an Outside Source Control Your Update Rate

Simultaneous Buffered Waveform Acquisition and Generation

**Part IV Getting Square with Digital I/O**

Things You Should Know about Digital I/O

When You Need It Now—Immediate Digital I/O

Shaking Hands with a Digital Partner

**Part V SCXI—Getting Your Signals in Great Condition**

Things You Should Know about SCXI

Hardware and Software Setup for Your SCXI System

Special Programming Considerations for SCXI

Common SCXI Applications

SCXI Calibration—Increasing Signal Measurement Precision

**Part VI Counting Your Way to High-Precision Timing**

Things You Should Know about Counters

Generating a Square Pulse or Pulse Trains

Measuring Pulse Width

Measuring Frequency and Period

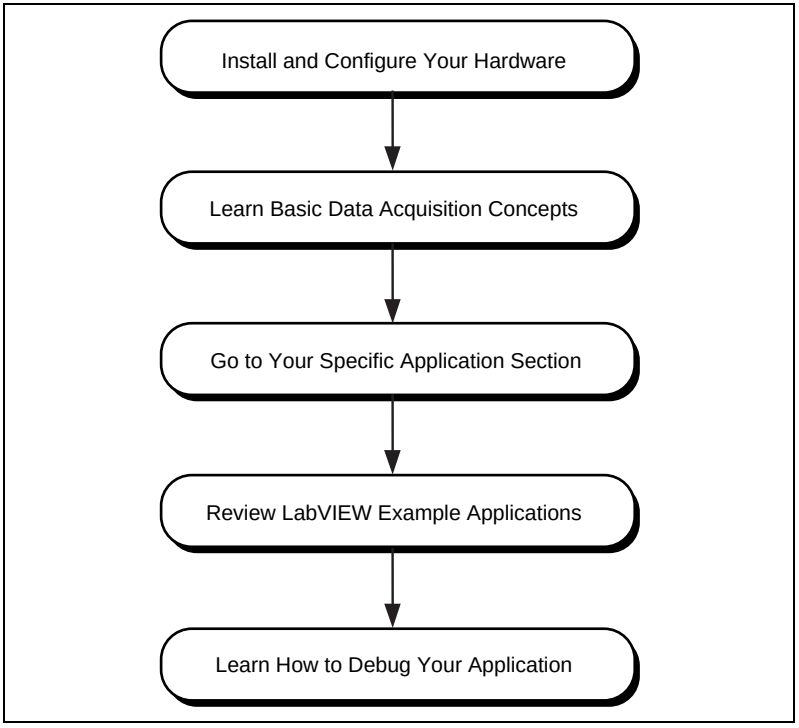
Counting Signal Highs and Lows

Dividing Frequencies

**Part VII Debugging Your Data Acquisition Application**

Debugging Techniques

If you already have started a LabVIEW DAQ application, please refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware*, to check your configuration. Refer to Part VII, *Debugging Your Data Acquisition Application*, for information on common errors. The following flowchart shows the steps to follow before running your application.


Install and Configure Your Hardware
Learn Basic Data Acquisition Concepts
Go to Your Specific Application Section
Review LabVIEW Example Applications
Learn How to Debug Your Application

1. **Install and Configure Your Hardware**—When you install LabVIEW, the program prompts you to have the data acquisition (DAQ) drivers installed. This manual guides you through setting up NI-DAQ software with your DAQ device and SCXI hardware. You should read any unique installation instructions for your platform in Chapter 2, *Installing and Configuring Your Data Acquisition Hardware*.
2. **Learn Basic Data Acquisition Concepts**—Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, shows you the location of DAQ example VIs; DAQ VI organization; VI parameter conventions; default and current value conventions; common VI parameter definitions; error handling; channel, port and counter addressing; limit settings; and data organization for analog applications.

3. **Go to Your Specific Application Section**—Chapter 4, *Where You Should Go Next*, shows you where to find information in this manual for your application.
4. **Review LabVIEW Example Applications**—The remaining chapters teach you basic concepts in analog input and output, digital I/O, counters, and SCXI. Each application section first lists example VIs, then describes the basic concepts needed to understand these example VIs. Whenever possible, you should have the VI open as you refer to these examples.
5. **Learn How to Debug Your Application**—Chapter 29, *Debugging Techniques*, describes the different ways you can debug your application. This chapter helps you troubleshoot for common programming errors.

# 2

# Installing and Configuring Your Data Acquisition Hardware

This chapter explains how to set up your system to use data acquisition with LabVIEW and your data acquisition hardware. The chapter contains hardware installation and configuration and software configuration instructions and some general information and techniques.

**Note** The LabVIEW installer prompts you to have the NI-DAQ driver software installed. All National Instruments data acquisition (DAQ) devices are packaged with NI-DAQ driver software. The version of NI-DAQ packaged with your DAQ device might be newer than the version installed by LabVIEW. You can determine the NI-DAQ version in LabVIEW by running the **Get DAQ Device Information VI**, located in **Functions»Data Acquisition»Calibration and Configuration**.

After installing LabVIEW and the NI-DAQ driver, follow the steps in Figure 2-1 to install your hardware and complete the software configuration. LabVIEW uses the software configuration information to recognize your hardware and to set default DAQ parameters.

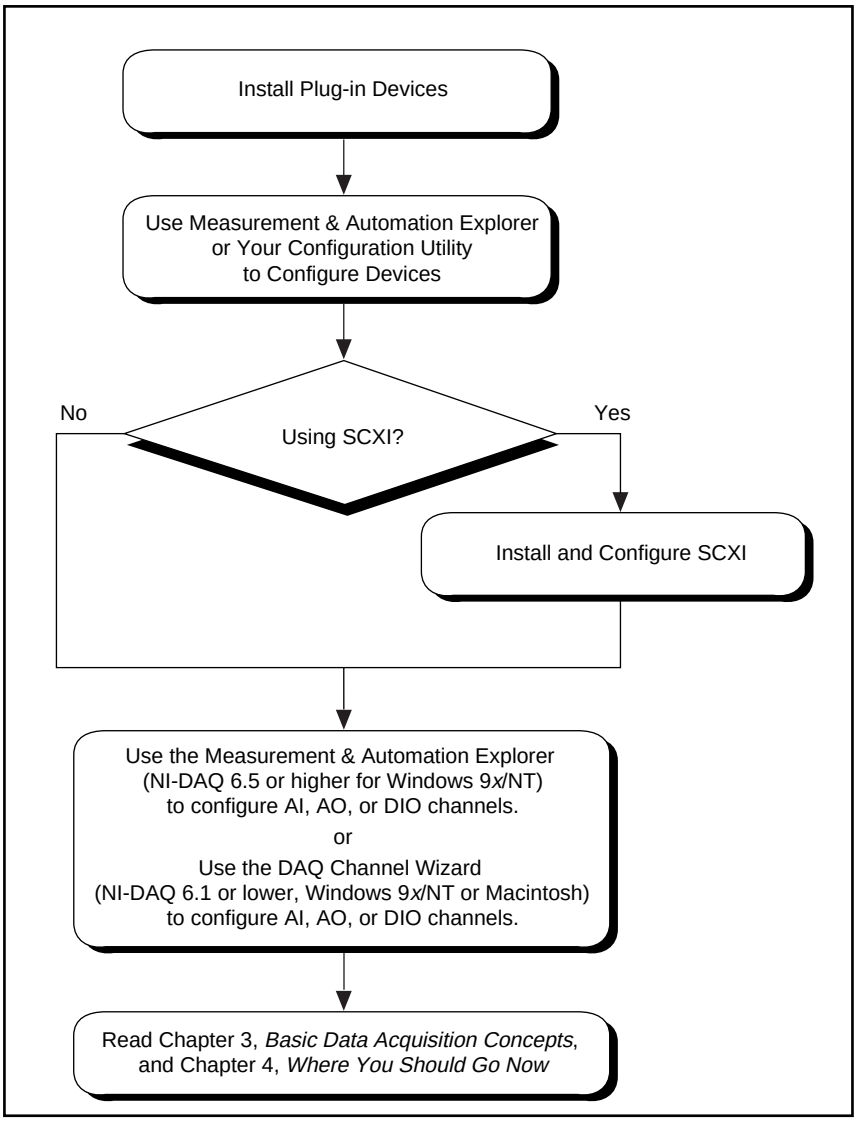

Install Plug-in Devices
Use Measurement & Automation Explorer or Your Configuration Utility to Configure Devices
No
Using SCXI?
Yes
Install and Configure SCXI
Use the Measurement & Automation Explorer (NI-DAQ 6.5 or higher for Windows 9x/NT) to configure AI, AO, or DIO channels.
or
Use the DAQ Channel Wizard (NI-DAQ 6.1 or lower, Windows 9x/NT or Macintosh) to configure AI, AO, or DIO channels.
Read Chapter 3, Basic Data Acquisition Concepts, and Chapter 4, Where You Should Go Now

**Figure 2-1.** Installing and Configuring DAQ Devices

NI-DAQ driver software provides LabVIEW with a high-level interface to DAQ devices and signal conditioning hardware.

Figure 2-2 shows the relationship between LabVIEW, NI-DAQ, and DAQ hardware.

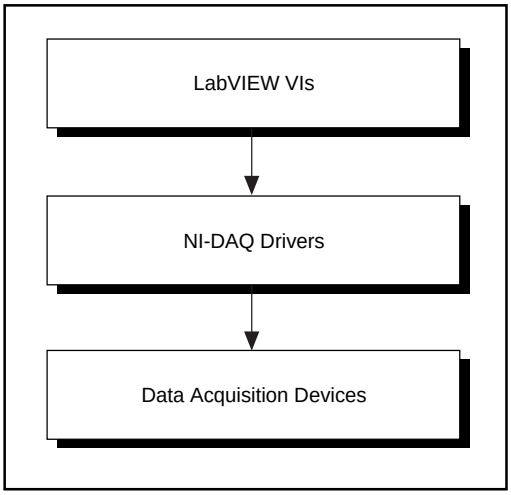



**Figure 2-2.** How NI-DAQ Relates to Your System and DAQ Devices

**(NI-DAQ 4.8.*x* for Macintosh)** NI-DAQ 4.8.*x* for the Macintosh device drivers are bundled in a single file that determines which drivers to load. When you restart your computer, this control panel driver, called `NI-DAQ`, determines which devices are installed in the system and loads their corresponding drivers. NI-DAQ uses its control panel settings to determine what Signal Conditioning eXtensions for Instrumentation (SCXI) hardware is configured and what the default device settings are for devices in the computer. If you use DMA, NI-DAQ also communicates with the NI-DMA/DSP for DMA services. When you install LabVIEW, the installer places both of these files on your hard drive.

**(NI-DAQ 6.*x* for Macintosh)** The NI-DAQ driver, called **NI-DAQ** is installed in the **National Instruments** folder in your Macintosh **Extensions** folder.

**(NI-DAQ 5.*x*, 6.*x* for Windows)** The NI-DAQ driver, `NIDAQ32.DLL`, is installed in your Windows `SYSTEM` or `SYSTEM32` directory.

# LabVIEW Data Acquisition Hardware Support

National Instruments periodically upgrades LabVIEW to add support for new DAQ hardware. To make sure this version of LabVIEW supports the hardware you use, refer to the following table.

**Table 2-1.** LabVIEW DAQ Hardware Support with NI-DAQ

| Device Type | NI-DAQ Version Support | | | | |
|---|---|---|---|---|---|
| | **Device** | **Windows 9*x*** | **Windows NT 4.0** | **Windows 2000** | **Macintosh** |
| PCI Bus Devices | NI 4060 for PCI | 6.1 and higher | 6.1 and higher | 6.6 and higher | N/A |
| | NI 4351 for PCI | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | NI 4551 for PCI | 6.1 and higher | 6.1 and higher | 6.6 and higher | N/A |
| | NI 4552 for PCI | 6.1 and higher | 6.1 and higher | 6.6 and higher | N/A |
| | NI 5102 for PCI | 5.0 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | NI 5401 for PCI | 6.5.2 and higher | 6.5.2 and higher | 6.6 and higher | N/A |
| | NI 5411 for PCI | 5.0 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | NI 5431 for PCI | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PCI-1200 | 4.9 and higher | 5.0 and higher | 6.6 and higher | 4.9 and higher |
| | PCI-4451 | 6.1 and higher | 6.1 and higher | 6.6 and higher | N/A |
| | PCI-4452 | 6.1 and higher | 6.1 and higher | 6.6 and higher | N/A |
| | PCI-4453 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | PCI-4454 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | PCI-6023E | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PCI-6024E | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PCI-6025E | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PCI-6031E | 5.1 and higher | 5.1 and higher | 6.6 and higher | 6.0 and higher |
| | PCI-6032E | 5.1 and higher | 5.1 and higher | 6.6 and higher | 6.0 and higher |
| | PCI-6033E | 5.1 and higher | 5.1 and higher | 6.6 and higher | 6.0 and higher |
| | PCI-6034E | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | PCI-6035E | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | PCI-6052E | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |

**Table 2-1.** LabVIEW DAQ Hardware Support with NI-DAQ (Continued)

| Device Type | NI-DAQ Version Support | | | | |
|---|---|---|---|---|---|
| | **Device** | **Windows 9*x*** | **Windows NT 4.0** | **Windows 2000** | **Macintosh** |
| PCI Bus Devices (Continued) | PCI-6071E | 5.1 and higher | 5.1 and higher | 6.6 and higher | 6.0 and higher |
| | PCI-6110E | 6.1 and higher | 6.1 and higher | 6.6 and higher | 6.1 and higher |
| | PCI-6111E | 6.1 and higher | 6.1 and higher | 6.6 and higher | 6.1 and higher |
| | PCI-6503 | 6.1 and higher | 6.1 and higher | 6.6 and higher | N/A |
| | PCI-6527 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | PCI-6601 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | PCI-6602 | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PCI-6608 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | PCI-6703 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | PCI-6704 | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PCI-6711 | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PCI-6713 | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PCI-7030[2] | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PCI-7040[2] | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PCI-DIO-32HS | 5.0 and higher | 5.0 and higher | 6.6 and higher | 6.0 and higher |
| | PCI-DIO-96 | 4.9 and higher | 5.0 and higher | 6.6 and higher | 4.9 and higher |
| | PCI-MIO-16E-1 | 5.0 and higher | 5.0 and higher | 6.6 and higher | 6.0 and higher |
| | PCI-MIO-16E-4 | 5.0 and higher | 5.0 and higher | 6.6 and higher | 6.0 and higher |
| | PCI-MIO-16XE-10 | 5.0 and higher | 5.0 and higher | 6.6 and higher | 6.0 and higher |
| | PCI-MIO-16XE-50 | 4.9 and higher | 5.0 and higher | 6.6 and higher | 4.9 and higher |
| PXI Combo Chassis | PXI-1010 | 6.0 and higher | 6.0 and higher | 6.6 and higher | N/A |
| | PXI-1011 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |

**Table 2-1.** LabVIEW DAQ Hardware Support with NI-DAQ (Continued)

| Device Type | NI-DAQ Version Support | | | | |
|---|---|---|---|---|---|
| | **Device** | **Windows 9*x*** | **Windows NT 4.0** | **Windows 2000** | **Macintosh** |
| PXI Bus Devices | NI 4060 for PXI | 6.1 and higher | 6.1 and higher | 6.6 and higher | N/A |
| | NI 4351 for PXI | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | NI 5102 for PXI | 6.0 and higher | 6.0 and higher | 6.6 and higher | N/A |
| | NI 5401 for PXI | 6.5.2 and higher | 6.5.2 and higher | 6.6 and higher | N/A |
| | NI 5411 for PXI | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | NI 5431 for PXI | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PXI-6011E | 5.1 and higher | 5.1 and higher | 6.6 and higher | N/A |
| | PXI-6025E | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PXI-6030E | 5.1 and higher | 5.1 and higher | 6.6 and higher | N/A |
| | PXI-6031E | 6.1 and higher | 6.1 and higher | 6.6 and higher | N/A |
| | PXI-6035E | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | PXI-6040E | 5.1 and higher | 5.1 and higher | 6.6 and higher | N/A |
| | PXI-6052E | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | PXI-6070E | 5.1 and higher | 5.1 and higher | 6.6 and higher | N/A |
| | PXI-6071E | 6.1 and higher | 6.1 and higher | 6.1 and higher | N/A |
| | PXI-6508 | 6.0 and higher | 6.0 and higher | 6.6 and higher | N/A |
| | PXI-6527 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | PXI-6533 | 5.1 and higher | 5.1 and higher | 6.6 and higher | N/A |
| | PXI-6602 | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PXI-6608 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | PXI-6703 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | PXI-6704 | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PXI-6711 | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PXI-6713 | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PXI-7030[2] | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | PXI-7040[2] | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |

**Table 2-1.** LabVIEW DAQ Hardware Support with NI-DAQ (Continued)

| Device Type | NI-DAQ Version Support | | | | |
|---|---|---|---|---|---|
| | **Device** | **Windows 9*x*** | **Windows NT 4.0** | **Windows 2000** | **Macintosh** |
| PC Card Devices | DAQCard-500 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | DAQCard-700 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | DAQCard-1200 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | DAQCard-516 | 4.9 and higher | 5.0 and higher | 6.6 and higher | 6.0 and higher |
| | DAQCard-6023E | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | DAQCard-6024E | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | DAQCard-6062E | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | DAQCard-6533 | 5.1 and higher | 5.1 and higher | 6.6 and higher | 6.0 and higher |
| | DAQCard-6723 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | DAQCard-AI-16E-4 | 4.9 and higher | 5.0 and higher | 6.6 and higher | 6.0 and higher |
| | DAQCard-AI-16XE-50 | 4.9 and higher | 5.0 and higher | 6.6 and higher | 6.0 and higher |
| | DAQCard-AO-2DC | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | DAQCard-DIO-24 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | NI 4050 for PCMCIA | 5.0 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | NI 4350 for PCMCIA | 5.1 and higher | 5.1 and higher | 6.6 and higher | N/A |
| | NI 5102 for PCMCIA | 5.0 and higher | 5.0 and higher | 6.6 and higher | N/A |
| External Devices | DAQPad-1200[1] | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | DAQPad-6020E (USB) | 6.5 and higher[3] | N/A | 6.6 and higher | N/A |
| | DAQPad-6052E (1394) | 6.6 and higher[3] | N/A | 6.6 and higher | N/A |
| | DAQPad-6070E (1394) | 6.5 and higher[3] | N/A | 6.6 and higher | N/A |
| | DAQPad-6507/6508 (USB) | 6.0 and higher[3] | N/A | 6.6 and higher | N/A |
| | DAQPad-6533 (1394) | 6.5 and higher[3] | N/A | 6.6 and higher | N/A |
| | DAQPad-6713 (1394) | 6.5 and higher[3] | N/A | 6.6 and higher | N/A |
| | DAQPad-MIO-16XE-50[1] | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | NI 4350 for USB | 6.0 and higher[3] | N/A | 6.6 and higher | N/A |
| | NI 5102 for USB | 6.0 and higher[3] | N/A | 6.6 and higher | N/A |

**Table 2-1.** LabVIEW DAQ Hardware Support with NI-DAQ (Continued)

| Device Type | NI-DAQ Version Support | | | | |
|---|---|---|---|---|---|
| | **Device** | **Windows 9*x*** | **Windows NT 4.0** | **Windows 2000** | **Macintosh** |
| AT Bus Devices | AT-A2150 | 4.8.5 to 4.9 | Contact NI | N/A | N/A |
| | AT-AI-16XE-10 | 4.9 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | AT-AO-6/10 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | AT-DIO-32F | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | AT-DIO-32HS | 5.0 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | AT-DSP2200 | 4.8.5 to 4.9 | Contact NI | N/A | N/A |
| | AT-MIO-16/16D | 4.8.5 to 6.1 | 5.0 to 6.1 | N/A | N/A |
| | AT-MIO-16DE-10 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | AT-MIO-16E-1 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | AT-MIO-16E-2 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | AT-MIO-16E-10 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | AT-MIO-16F-5 | 4.8.5 to 6.1 | 5.0 to 6.1 | N/A | N/A |
| | AT-MIO-16X | 4.8.5 to 6.1 | 5.0 to 6.1 | N/A | N/A |
| | AT-MIO-16XE-10 | 4.9 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | AT-MIO-16XE-50 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | AT-MIO-64E-3 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | AT-MIO-64F-5 | 4.8.5 to 6.1 | 5.0 to 6.1 | N/A | N/A |
| | NI 5102 for ISA | 5.0 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | NI 5411 for ISA | 5.0 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | NI 5431 for ISA | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | EISA-A2000 | 4.8.5 to 4.9 | Contact NI | N/A | N/A |

**Table 2-1.** LabVIEW DAQ Hardware Support with NI-DAQ (Continued)

| Device Type | NI-DAQ Version Support | | | | |
|---|---|---|---|---|---|
| | **Device** | **Windows 9*x*** | **Windows NT 4.0** | **Windows 2000** | **Macintosh** |
| PC/XT Bus Devices | NI 4350 for ISA | 5.1 and higher | 5.1 and higher | 6.6 and higher | N/A |
| | PC-516 | 4.9 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | PC-AO-2DC | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | PC-DIO-24 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | PC-DIO-24PnP | 6.0 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | PC-DIO-96 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | PC-DIO-96PnP | 6.0 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | PC-LPM-16 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | PC-LPM-16PnP | 4.9 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | PC-OPDIO-16 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | PC-TIO-10 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| Lab Series Devices | Lab-PC+ | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | Lab-PC-1200 | 4.9 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | Lab-PC-1200AI | 4.9 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | Lab-LC | N/A | N/A | N/A | 4.8 to 4.9 |
| | Lab-NB | N/A | N/A | N/A | 4.8 to 4.9 |
| NEC Devices | NEC-AI-16E-4 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | NEC-AI-16XE-50 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | NEC-MIO-16E-4 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | NEC-MIO-16XE-50 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |

**Table 2-1.** LabVIEW DAQ Hardware Support with NI-DAQ (Continued)

| Device Type | NI-DAQ Version Support | | | | |
|---|---|---|---|---|---|
| | **Device** | **Windows 9*x*** | **Windows NT 4.0** | **Windows 2000** | **Macintosh** |
| NuBus Devices | NB-A2000 | N/A | N/A | N/A | 4.8 to 4.9 |
| | NB-A2150 | N/A | N/A | N/A | 4.8 to 4.9 |
| | NB-A2100 | N/A | N/A | N/A | 4.8 to 4.9 |
| | NB-AO-6 | N/A | N/A | N/A | 4.8 to 4.9 |
| | NB-DIO-24 | N/A | N/A | N/A | 4.8 to 4.9 |
| | NB-DIO-32F | N/A | N/A | N/A | 4.8 to 4.9 |
| | NB-DIO-96 | N/A | N/A | N/A | 4.8 to 4.9 |
| | NB-DMA-8-G | N/A | N/A | N/A | 4.8 to 4.9 |
| | NB-DMA2800 | N/A | N/A | N/A | 4.8 to 4.9 |
| | NB-MIO-16 | N/A | N/A | N/A | 4.8 to 4.9 |
| | NB-MIO-16X | N/A | N/A | N/A | 4.8 to 4.9 |
| | NB-TIO-10 | N/A | N/A | N/A | 4.8 to 4.9 |
| VXI Carriers and Modules | VXI-AO-48XDC | 4.9 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | VXI-DIO-128 | 4.9 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | VXI-MIO-64E-1 | 4.9 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | VXI-MIO-64XE-10 | 4.9 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | VXI-SC-1000 | 5.0 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | VXI-SC-1102 | 5.0 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | VXI-SC-1102B | 5.1 and higher | 5.1 and higher | 6.6 and higher | N/A |
| | VXI-SC-1102C | 5.1 and higher | 5.1 and higher | 6.6 and higher | N/A |
| | VXI-SC-1104 | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | VXI-SC-1150 | 5.0 and higher | 5.0 and higher | 6.6 and higher | N/A |
| SCXI Chassis | SCXI-1000 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1000DC | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1001 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-2000 | 4.9 and higher | 5.0 and higher | 6.6 and higher | N/A |

**Table 2-1.** LabVIEW DAQ Hardware Support with NI-DAQ (Continued)

| Device Type | NI-DAQ Version Support | | | | |
|---|---|---|---|---|---|
| | **Device** | **Windows 9*x*** | **Windows NT 4.0** | **Windows 2000** | **Macintosh** |
| SCXI Modules | SCXI-1100 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1101 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCXI-1102 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1102B | 5.1 and higher | 5.1 and higher | 6.6 and higher | 6.0 and higher |
| | SCXI-1102C | 5.1 and higher | 5.1 and higher | 6.6 and higher | 6.0 and higher |
| | SCXI-1104 | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | SCXI-1112 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCXI-1120 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1120D | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1121 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1122 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1124 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1125 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCXI-1126 | 6.0 and higher | 6.0 and higher | 6.6 and higher | 6.0 and higher |
| | SCXI-1127 | 6.5.2 and higher | 6.5.2 and higher | 6.6 and higher | N/A |
| | SCXI-1140 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1141 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1142 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCXI-1143 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCXI-1160 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1161 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1162 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1162HV | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1163 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1163R | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SCXI-1200[1] | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | N/A |
| | SCXI-2400 | 4.9 and higher | 5.0 and higher | 6.6 and higher | N/A |

**Table 2-1.** LabVIEW DAQ Hardware Support with NI-DAQ (Continued)

| Device Type | NI-DAQ Version Support | | | | |
|---|---|---|---|---|---|
| | **Device** | **Windows 9*x*** | **Windows NT 4.0** | **Windows 2000** | **Macintosh** |
| Accessories | AMUX-64T | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SC-2040 | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SC-2042-RTD | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| | SC-2043-SG | 4.8.5 and higher | 5.0 and higher | 6.6 and higher | 4.8 and higher |
| Signal Conditioning Components (SCC) and Carriers | SC-2345 | 6.0 and higher | 6.0 and higher | 6.6 and higher | 6.0 and higher |
| | SCC-A10 | 6.0 and higher | 6.0 and higher | 6.6 and higher | 6.0 and higher |
| | SCC-AI01 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCC-AI02 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCC-AI03 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCC-AI04 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCC-AI05 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCC-AI06 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCC-AI07 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCC-AI13 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCC-AI14 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCC-CI20 | 6.0 and higher | 6.0 and higher | 6.6 and higher | 6.0 and higher |
| | SCC-DI01 | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | SCC-DO01 | 6.5 and higher | 6.5 and higher | 6.6 and higher | N/A |
| | SCC-FT01 | 6.0 and higher | 6.0 and higher | 6.6 and higher | 6.0 and higher |
| | SCC-LP01 | 6.1 and higher | 6.1 and higher | 6.6 and higher | 6.1 and higher |
| | SCC-LP02 | 6.1 and higher | 6.1 and higher | 6.6 and higher | 6.1 and higher |
| | SCC-LP03 | 6.1 and higher | 6.1 and higher | 6.6 and higher | 6.1 and higher |

**Table 2-1.** LabVIEW DAQ Hardware Support with NI-DAQ (Continued)

| Device Type | NI-DAQ Version Support | | | | |
|---|---|---|---|---|---|
| | **Device** | **Windows 9*x*** | **Windows NT 4.0** | **Windows 2000** | **Macintosh** |
| Signal Conditioning Components (SCC) and Carriers (Continued) | SCC-PWR01 | 6.0 and higher | 6.0 and higher | 6.6 and higher | 6.0 and higher |
| | SCC-PWR02 | 6.0 and higher | 6.0 and higher | 6.6 and higher | 6.0 and higher |
| | SCC-PWR03 | 6.0 and higher | 6.0 and higher | 6.6 and higher | 6.0 and higher |
| | SCC-SG01 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCC-SG02 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCC-SG03 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCC-SG04 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCC-SG11 | 6.6 and higher | 6.6 and higher | 6.6 and higher | N/A |
| | SCC-TC01 | 6.0 and higher | 6.0 and higher | 6.6 and higher | 6.0 and higher |
| | SCC-TC02 | 6.0 and higher | 6.0 and higher | 6.6 and higher | 6.0 and higher |

[1] The DAQPad-MIO-16XE-50 and DAQPad-1200 do not work with NEC PC-9800 Series computers. The SCXI-1200 works with NEC PC-9800 Series computers ONLY when used with Remote SCXI.

[2] Requires LabVIEW/RT.

[3] USB and IEEE-1394 devices are supported only on Windows 98 and Windows 2000. There is no support under Windows 95 and Windows NT 4.0.

For more information regarding hardware support for LabVIEW, refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Function and VI Reference Manual*, or the LabVIEW *Online Reference*, by selecting **Help»Online Reference**.

# Installing and Configuring Your National Instruments Device

Some DAQ devices have jumpers to set analog input polarity, input mode, analog output reference, and so on. Before you install your device, check your hardware user manuals to see if your device has jumpers and how to change its settings. You then can determine whether you need to change any jumper settings. Record any jumper settings that you change so that you can enter the information correctly in the configuration utility.

The next step depends on what version of NI-DAQ you have. Go to the appropriate section below to continue the configuration of your devices.

## Installing and Configuring Your DAQ Device Using NI-DAQ 5.*x*, 6.*x*

You can refer to the online help for specific instructions on how to install and configure your DAQ device. If you are using Windows 9*x*/NT with NI-DAQ 6.5 or higher, refer to the help file for Measurement & Automation Explorer by selecting **Help»Help Topics»DAQ Help**. If you are using an earlier version of NI-DAQ for Windows, you can find the help file in **Start»Programs»LabVIEW»NI-DAQ Configuration Utility Help**. If you are using a Macintosh, you can find the help file in the **Help** menu of the NI-DAQ Configuration Utility.

## Configuring Your DAQ Device Using NI-DAQ 4.8.*x* on the Macintosh

After you check and record your jumper settings, turn off your computer and insert your National Instruments devices.

NI-DAQ

Turn your computer back on. You can find `NI-DAQ` in your `control panels` folder. The NI-DAQ icon looks like the one shown to the left. Double-click on this icon to launch NI-DAQ.

When you launch the program, NI-DAQ displays a list of all of the devices in your computer. Each device has a list of attributes, as shown in Figure 2-3. The number specified in the device line is the logical device number that NI-DAQ assigned to the device. You use this number in LabVIEW as the device number to select that device for any operation.

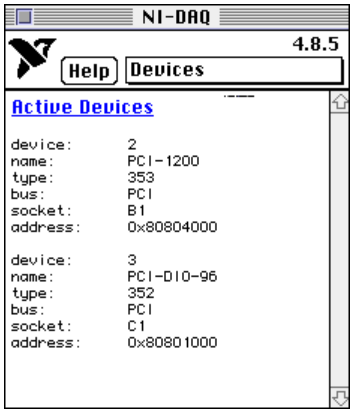



**Figure 2-3.** NI-DAQ Device Window Listing

Now show the Device Configuration window by selecting the **Device Configuration** option from the menu as shown in Figure 2-4.

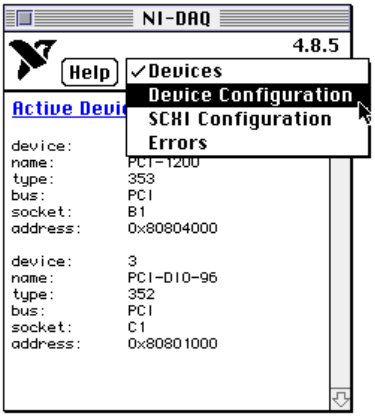



**Figure 2-4.** Accessing the Device Configuration Window in NI-DAQ

Figure 2-5 shows the NI-DAQ Device Configuration window. When you are in the Device Configuration window of the utility, you can edit the default settings for parameters, such as analog input polarity and range on a per-device basis. If you are using AMUX-64T or signal conditioning devices with your DAQ device, select the appropriate device using the **Accessories** menu. LabVIEW uses these settings when initializing the device instead of the default settings listed in the descriptions of the hardware configuration VIs. (You can use these VIs to change any setting recorded by NI-DAQ.) When you click on the name of the device, NI-DAQ displays the I/O connector for the device, as shown in Figure 2-5.

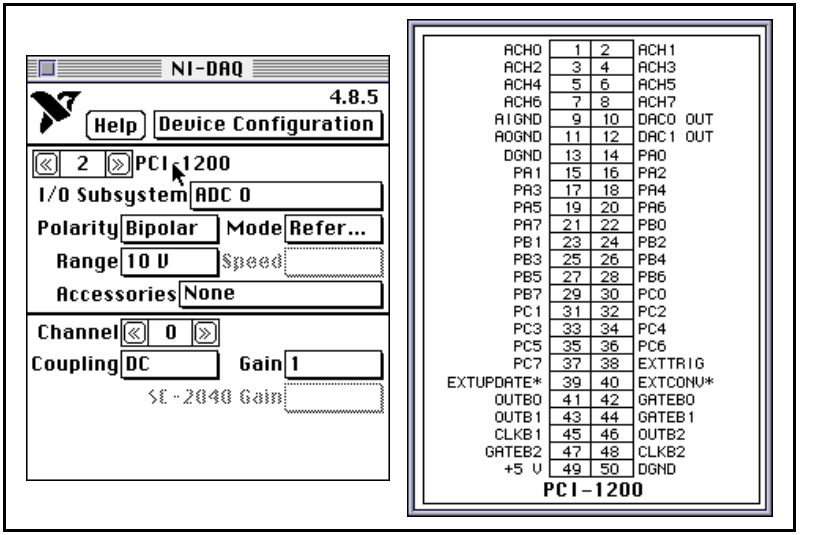



**Figure 2-5.** Device Configuration and I/O Connector Windows in NI-DAQ

You also can find additional information by clicking the **Help** button. If at any time during configuration you need to view a list of the LabVIEW DAQ error codes and their meanings, you can do so by clicking on the NI-DAQ menu bar, located to the right of the **Help** button, and choosing **Errors**.

**Note** Some DAQ devices, such as the Lab-NB and NB-MIO-16 devices, require hardware jumper changes in addition to software configuration. Consult your DAQ device hardware reference manual for more information.

## Installing and Configuring Your SCXI Chassis

The following section describes the procedures for installing and configuring your SCXI chassis.

### Hardware Configuration

Your SCXI hardware kit includes the *Getting Started with SCXI* manual, which contains detailed instructions for assembling your SCXI system, module jumper settings, cable assemblies, and terminal blocks.

The following are the basic steps you must complete to assemble your SCXI system:

1. Check the jumpers on your modules. Generally, you will leave the jumpers in their default positions. However, the *Getting Started with SCXI* manual contains a section for each module type that lists cases where you might want to change the jumper settings.
2. Turn off the chassis power. Plug in your modules through the front of the chassis. You can put the modules in any slot. For simplicity, start with slot 1 on the left side of the chassis and move right with each additional module. Be sure to tightly screw the modules into the chassis frame.
3. If you are using an SCXI-1180 feedthrough panel, you must install the SCXI-1180 in the slot immediately to the right of the module that you will cable to the DAQ device. Otherwise, the cable connectors might not fit together conveniently.
4. If you have more than one chassis, select a unique jumpered address for each additional chassis by using the jumpers directly behind the front panel of the chassis.
5. Plug the appropriate terminal blocks into the front of each module and screw them tightly into the chassis frame.
6. If you are using a DAQ device in your computer to control your SCXI chassis, connect the mounting bracket of the SCXI-134*x* (where *x* is a number) cable assembly to the back of one of the modules and screw it into the chassis frame. Connect the other end of the cable to the DAQ device in your computer. In multiplexed mode, you need to connect only one module to the DAQ device. In most cases, it does not matter which module you cable. The following are two special cases where you must cable a specific module to the device:
   a. If you use SCXI-1140 modules with other types of modules, you need to cable one of the SCXI-1140 modules to the DAQ device.
   b. If you use analog input modules and other types of modules, you need to cable one of the analog input modules to the DAQ device.
7. Turn on your chassis power.

Refer to the *Getting Started with SCXI* manual for more information about related topics, such as multichassis cabling.

## NI-DAQ 5.*x*, 6.*x* Software Configuration

You can refer to the online help for specific instructions on how to install and configure your DAQ device. If you are using Windows 9*x*/NT with NI-DAQ 6.5 or higher, refer to the help file for Measurement & Automation Explorer by selecting **Help»Help Topics»DAQ Help**. If you are using an earlier version of NI-DAQ for Windows, you can find the help file in **Start»Programs»LabVIEW»NI-DAQ Configuration Utility Help**. If you are using a Macintosh, you can find the help file in the **Help** menu of the NI-DAQ Configuration Utility.

## NI-DAQ 4.8.*x* for Macintosh Software Configuration

To use SCXI with LabVIEW and NI-DAQ 4.8.*x*, you must enter the configuration for each SCXI chassis using NI-DAQ. Select **SCXI Configuration** in the NI-DAQ menu bar to bring up the SCXI Configuration window as shown in Figure 2-6.

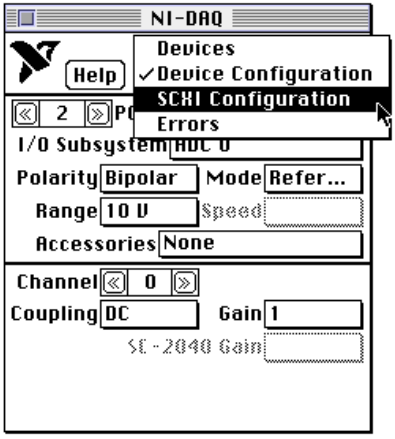



**Figure 2-6.** Accessing the NI-DAQ SCXI Configuration Window

Figure 2-7 shows NI-DAQ with the SCXI Configuration window selected.

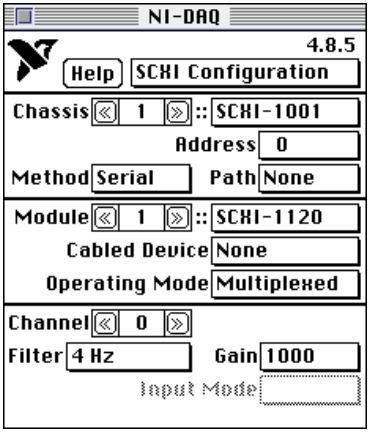



**Figure 2-7.** SCXI Configuration Window in NI-DAQ

Follow these steps to configure your SCXI chassis and modules:

1. Leave the **Chassis** control set to `1` if you have only one chassis. You will use this number to access the SCXI chassis from your application. If you have multiple chassis, advance the **Chassis** control to configure each subsequent chassis.
2. Select the appropriate chassis type. This activates the remaining fields on the panel.
3. If you have only one chassis, leave the **Address** field and the address jumpers on your SCXI chassis set to `0`. If you have additional chassis, you must select a unique hardware-jumpered address for each chassis and enter it in the **Address** field.
4. Leave the **Method** set to `Serial`, which means that LabVIEW communicates with the chassis serially using a DIO port of the plug-in DAQ device. The **Path** automatically sets itself to the device number of the appropriate DAQ device when you enter the **Cabled Device** information in step 5b.

5. Enter the configuration for each slot in the chassis. The fields in the bottom two sections of the window reflect the settings for the selected **Module** number. Refer to your SCXI chassis hardware manual to determine how the slots in a chassis are numbered. You must set the following fields for each SCXI module you install:

   a. **Module type**—Select the correct module type for the module installed in the current slot. If the current slot does not have a module, leave this field set to `None` and advance the **Module** number to the next slot.

   b. **Cabled Device**—If the module in the current slot is *directly cabled* to a DAQ device in your computer, set this field to the device number of that DAQ device. Leave the **Cabled Device** field set to `None` if the module in the current slot is not directly cabled to a DAQ device. If you are operating your modules in multiplexed mode, you only need to cable one module in each chassis to your DAQ device. If you are not using multiplexed mode, refer to the *SCXI Operating Modes* section of Chapter 19, *Hardware and Software Setup for Your SCXI System*, for instructions about module cabling.

   c. **Operating Mode**—The system defaults to the multiplexed operating mode, which is recommended for almost all SCXI applications. The operating modes available for each SCXI module type are described in the *SCXI Operating Modes* section of Chapter 19, *Hardware and Software Setup for Your SCXI System*.

      If the module is an analog input module, enter the gain and filter settings for each channel in the bottom section of the window. The system disables the **Channel** control for any modules that use only one gain and filter setting for the entire module.

## Configuring Your Channels in NI-DAQ 5.*x*, 6.*x*

Once you install and configure your hardware, you can configure your channels. LabVIEW DAQ software includes a channel configuration application, the DAQ Channel Wizard, you can use to configure the analog and digital channels on your DAQ device—DAQ plug-in boards, standalone DAQ products, or SCXI modules. In NI-DAQ 6.5 or higher, the DAQ Channel Wizard is integrated into Measurement & Automation Explorer. In NI-DAQ 5.*x* only analog input channels can be configured. The DAQ Channel Wizard helps you define the physical quantities you are measuring or generating on each DAQ hardware channel by querying for information about the physical quantity being measured, the sensor or actuator being used, and the associated DAQ hardware.

As you configure channels in the DAQ Channel Wizard, you give each channel configuration a unique name that is used when addressing your channels in LabVIEW. The channel configurations you define are saved in a file that instructs the NI-DAQ driver how to scale and process each DAQ channel by its name. You can simplify the programming required to measure your signal by using the DAQ Channel Wizard to configure your channels.

You can refer to the online help for specific instructions on how to install and configure your DAQ device. If you are using Windows 9*x*/NT with NI-DAQ 6.5 or higher, refer to the help file for Measurement & Automation Explorer by selecting **Help»Help Topics»DAQ Help**. If you are using an earlier version of NI-DAQ for Windows, you can find the help file in **Start»Programs»LabVIEW»NI-DAQ Configuration Utility Help**. If you are using a Macintosh, you can find the help file in the **Help** menu of the NI-DAQ Configuration Utility.

Refer to the *Channel Name Addressing* section of Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, for information about how to use your named channels in LabVIEW.

Now that you have successfully installed and configured your DAQ hardware for LabVIEW, read Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, for more information about data acquisition with LabVIEW.

# 3

# Basic LabVIEW Data Acquisition Concepts

This chapter explains how data acquisition works with LabVIEW. Before you start building your data acquisition (DAQ) application, you should know some of the following basic LabVIEW DAQ concepts:

- Location of common DAQ examples
- Locating the DAQ VIs in LabVIEW
- DAQ VI organization
- VI parameter conventions
- Common DAQ VI parameters
- Default and current value conventions
- Error handling
- Channel, port, and counter addressing
- Limit settings
- Data organization for analog applications

If you do not already understand basic programming concepts in LabVIEW, refer to the *LabVIEW User Manual* or the *G Programming Reference Manual* for help with programming in LabVIEW.

## Location of Common DAQ Examples

The DAQ examples address many common applications involving data acquisition in LabVIEW. You can find these examples in `labview\examples\daq`.

**(NI-DAQ 5.*x*, 6.*x*)** There are two ways to locate DAQ examples. One way is to run the DAQ Solution Wizard. The other way is to run the Search Examples Online Help, available in LabVIEW version 5.0 and higher. You can launch either tool from the LabVIEW launch screen.

The following list briefly describes the VI libraries (designated by the `.llb` file extension) and directories located in the `daq` directory.

| | |
|---|---|
| `accessories` | Folder containing VIs that work with specific accessories. |
| `anlogin` | Folder containing VIs that perform analog input and VIs that write or stream acquired data to disk. |
| `anlogout` | Folder containing VIs that generate single values or multiple values (waveforms) to output through analog channels. |
| `anlog_io` | Folder containing VIs for analog I/O control loops and simultaneous analog input and output. |
| `counter` | Folder containing VIs that perform various counter/timer operations. |
| `digital` | Folder containing VIs that perform immediate digital I/O and digital handshaking. |
| `scxi` | Folder containing VIs for use with specific SCXI modules. |
| `solution` | Folder containing a variety of ready-to-run application VIs. |
| `run_me.llb` | Library containing VIs that perform basic operations concerning analog I/O, digital I/O, and counters. |

Each chapter in this manual teaches the basic concepts behind several of the DAQ examples. For a brief description of any example, open the example VI and choose **Windows»Show VI Info** for a text description of the example. You also can choose **Help»Show Help** to open the **Help** window.

# Locating the Data Acquisition VIs in LabVIEW

You can find the Data Acquisition VIs in the **Functions** palette from your block diagram in LabVIEW. When you put your cursor over each of the icons in the **Functions** palette, LabVIEW displays the palette name you are about to access at the top of the **Functions** palette. You can find the Data Acquisition icon near the bottom of the **Functions** palette, as shown in Figure 3-1.

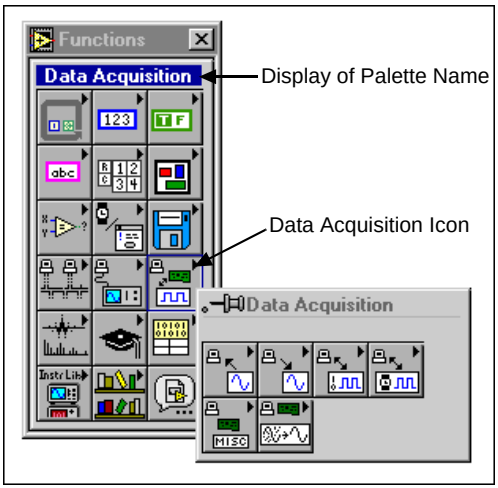



**Figure 3-1.** Accessing the Data Acquisition Palette

The **Data Acquisition** palette contains six subpalette icons that take you to the different classes of DAQ VIs. Figure 3-2 shows what each of the icons in the **Data Acquisition** palette means.

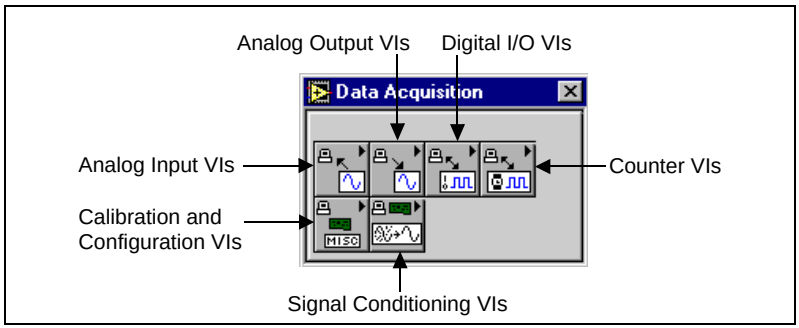



**Figure 3-2.** Data Acquisition VIs Palette

# DAQ VI Organization

In most of the DAQ VI subpalettes, the VIs are arranged in different levels according to their functionality. You can find some of the following four levels of DAQ VIs within the DAQ VI subpalettes:

- Easy VIs
- Intermediate VIs
- Utility VIs
- Advanced VIs

A good example of a palette that contains all of the available levels of DAQ VIs is the **Analog Input** palette. Figure 3-3 shows this palette.

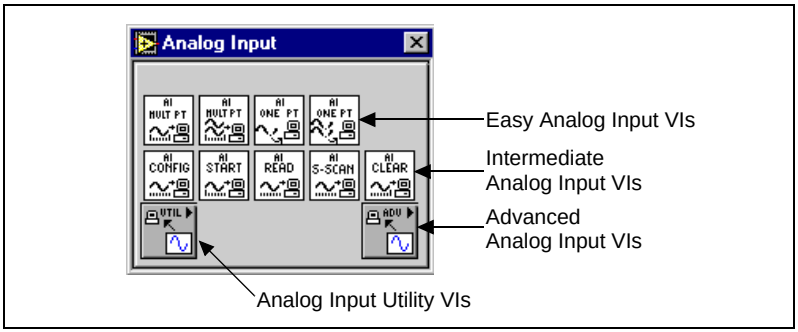



**Figure 3-3.** Analog Input VI Palette Organization

## Easy VIs

The Easy VIs perform simple DAQ operations and are typically the first row of VIs in the DAQ palettes. You can run these VIs from the front panel or use them as subVIs in basic applications.

You only need one Easy VI to perform each basic DAQ operation. Unlike intermediate- and advanced-level VIs, Easy VIs automatically alert you to errors with a dialog box that allows you to stop the execution of the VI or to ignore the error.

The Easy VIs usually are composed of Intermediate VIs, which are in turn composed of Advanced VIs. The Easy VIs provide a basic interface with only the most commonly used inputs and outputs. For more complex applications, you should use the intermediate- or advanced-level VIs for more functionality and better performance.

Refer to your particular type of VI in the *LabVIEW Function and VI Reference Manual* for specific VI information, or refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**.

## Intermediate VIs

The Intermediate VIs have more hardware functionality and efficiency in developing your application than the Easy VIs. The Intermediate VIs contain groups of Advanced VIs, but they use fewer parameters and do not have some of the more advanced capabilities.

Intermediate VIs give you more control over error-handling than the Easy VIs. With each VI, you can check for errors or pass the error cluster on to other VIs.

**Note** Most LabVIEW data acquisition examples shown in this manual are based on the Intermediate VIs. You can find these example VIs in the `examples` folder.

## Utility VIs

The Utility VIs, found in many of the LabVIEW DAQ palettes, are also intermediate-level VIs and thus have more hardware functionality and efficiency in developing your application than the Easy VIs.

## Advanced VIs

The Advanced VIs are the lowest-level interface to the DAQ driver. Very few applications require the use of the Advanced VIs. Use the Advanced VIs when the Easy or Intermediate VIs do not have the inputs necessary to control an unusual DAQ function. Advanced VIs return the greatest amount of status information from the DAQ driver. This manual primarily focuses on applications using the Easy or Intermediate VIs.

# VI Parameter Conventions

In each LabVIEW DAQ VI front panel or Help window, the appearance of the control and indicator labels denote the importance of that parameter. Control and indicator names shown in bold are required and must be wired to a node on the block diagram for your application to run. Parameter names that appear in plain text are optional and are not necessary for your program to run. You rarely need to use the parameters with labels in square brackets ([ ]). Keep in mind that these conventions apply only to the information in the Help window and on the front panel. Both this manual and the *LabVIEW Function and VI Reference Manual* list parameter names

in bold to distinguish them from other elements of the text. Default input values appear in parentheses to the right of the parameter names.

Figure 3-4 illustrates these Help window parameter conventions for the AI Read One Scan VI. As the window text for this VI indicates, you must wire the **device** (if you are not using channel names), **channels**, **error in**, and **iteration** input parameters and the **scaled data** and **error out** output parameters. To pass error information from one VI to another, connect the **error out** cluster of the current VI to the **error in** cluster of the next VI. The **coupling & input config**, **input limits**, **output units**, and **number of AMUX boards** input parameters and the **binary data** output parameter are optional.

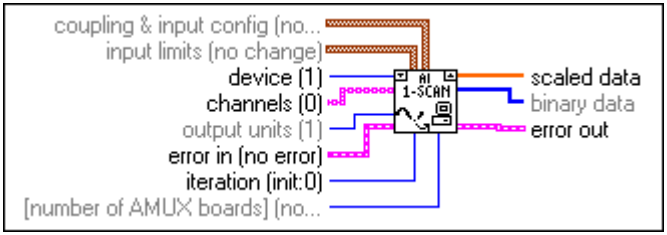



**Figure 3-4.** LabVIEW Help Window Conventions

# Default and Current Value Conventions

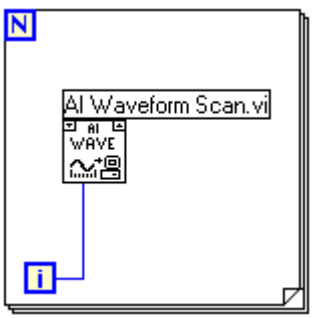
To use the DAQ VIs, you should know the difference between a default input, a default setting, and a current setting. A *default input* is the default value of a front panel control. When you do not wire an input to a terminal of a VI, the default input for the control associated with that terminal passes to the driver. In the **Help** window, default inputs appear in parentheses to the right of the parameter names. A *default setting* is a default parameter value recorded in the driver. The *current setting* is the value of a control at any given moment. The default setting of a control becomes the current setting and remains so until you change the value of the control.

In many cases, a control input defaults to a certain value (most often 0), which means you can use the current setting. For example, the default input for a parameter may be `do not change the current setting`, and the current setting may be `no AMUX-64T boards`. If you change the value of such a parameter, the new value becomes the current setting.

# Common DAQ VI Parameters

The **device** input on analog I/O, digital I/O, and counter VIs specifies the number assigned to your DAQ device in the DAQ configuration software. Your software assigns a unique number to each DAQ device. The **device** parameter usually appears as an input to the configuration VIs. Another common configuration VI output, **task ID**, assigns your specific I/O operation and device a unique number that identifies it throughout your program flow. The **task ID** can also contain group information about the channels and gain used in your operation.

Some DAQ VIs perform either the device configuration or the I/O operation, while other DAQ VIs perform both configuration and the operation. The VIs that handle both functions have an **iteration** input. When your VI has the **iteration** set to `0`, LabVIEW configures the DAQ device and then performs the specific I/O operation. For iteration values greater than 0, LabVIEW uses the existing configuration to perform the I/O operation. You can improve the performance of your application by not configuring the DAQ device every time an I/O operation occurs. Typically, you should wire the **iteration** input to an iteration terminal in a loop as shown in the following illustration.




Wiring the **iteration** input this way means the device is only configured on the first I/O operation. Subsequent I/O operations use the existing configuration.

# Error Handling

Each Easy VI contains an error handling VI. A dialog box appears immediately if an error occurs in an Easy VI.

Every Intermediate and Advanced VI contains an **error in** input cluster and an **error out** output cluster, as shown in Figure 3-5. The clusters contain a Boolean that indicates whether an error occurred, the **code** for the error, and **source** or the name of the VI that returned the error. If **error in** indicates an error, the VI passes the error information to **error out** and does not execute any DAQ functions.

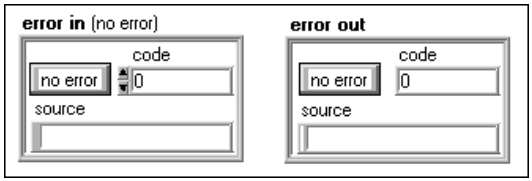



**Figure 3-5.** LabVIEW Error In Input and Error Out Output Error Clusters

For more information on error handling, refer to Part VII, *Debugging Your Data Acquisition Application*, in this manual.

# Channel, Port, and Counter Addressing

The Analog Input and Analog Output VIs have a **channel list** parameter where you can specify the channels from which the VIs read or to which they write. The Digital Input and Output VIs have a similar parameter, called **digital channel list** and the equivalent value is called **counter list** for the Counter VIs.

**Note** To simplify the explanation of channel addressing concepts, the **channel list**, **digital channel list**, and **counter list** parameters are all referred to as **channel list** in this section. Any special exceptions for these parameters are noted.

Each channel you specify in the **channel list** becomes a member of a group. For each group, you can acquire or generate data on the channels listed in the group. VIs scan (during acquisition) or update (during generation) the channels in the same order they are listed. To erase a group, pass an empty **channel list** and the group number to the VI or assign a new **channel list** to the group. Changing groups can only be done at the Advanced VI level. Refer to the *LabVIEW Function and VI Reference Manual* or the LabVIEW

*Online Reference*, by selecting **Help»Online Reference**, for more information.

## Channel Name Addressing

If you use the DAQ Channel Wizard to configure your analog and digital channels, you can address your channels by name in the **channel list** parameter in LabVIEW. **channel list** can be an array of strings or, as with the Easy VIs, a scalar string control, as shown in Figure 3-6. If you have a **channel list** array, you can use one channel entry per array element, specify the entire list in a single element, or use any combination of these two. If you enter multiple channel names in **channel list**, all of the channels in the list must be configured for the same DAQ device. If you configure channels with names of `temperature` and `pressure`, both of which are measured by the same DAQ device, you can specify a list of channels in a single element by separating them by commas—for example `temperature,pressure`. In specifying channel names, spelling and spaces are important, but case is not.

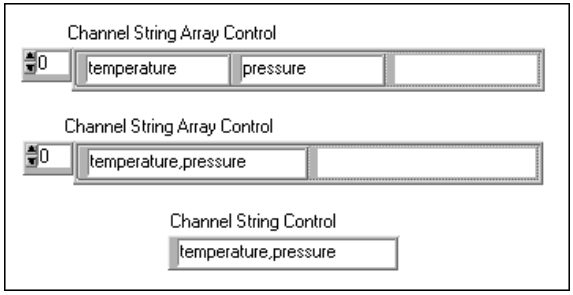



**Figure 3-6.** Channel String Controls

Using channel names, you do not need to wire the **device**, **input limits**, or **input config** input parameters. The **device** input is always ignored by LabVIEW when using channel names. LabVIEW configures your hardware in terms of your channel configuration.

**Tip** Unless you need to overwrite your channel name configuration, do not wire **input limits** or **input config**. Allow LabVIEW to set them up for you. In addition, LabVIEW orders and pads the channels specified in **channel list** for you as needed according to any special device requirements.

## Channel Number Addressing

If you are not using channel names to address your channels, you can address your channels by channel numbers in the **channel list** parameter. The **channel list** can be an array of strings or, as with the Easy VIs, a scalar string control. If you have a **channel list** array, you can use one channel entry per array element, specify the entire list in a single element, or use any combination of these two methods. For instance, if `0`, `1`, and `2` are your channels, you can specify a list of channels in a single element by separating the individual channels by commas—for example, `0, 1, 2`. Or, because `0` refers to the first channel in a consecutive channel range and `2` refers to the last channel, you can specify the range by separating the first and last channels with a colon—for example, `0:2`.

Some Easy and Advanced Digital VIs and Intermediate Counter VIs only need one port or counter to be specified. For more information, refer to the *LabVIEW Function and VI Reference Manual* or the LabVIEW *Online Reference*. You also can choose **Help**»**Show Help** and put your cursor on the VI to view the VI Help window for the VI you intend to use.

LabVIEW recognizes three types of analog channels on a DAQ device: onboard, AMUX-64T, and SCXI channels. It recognizes two types of digital ports and counters: onboard and SCXI. This section describes addressing onboard channels, ports, and counters. AMUX-64T addressing is described later in Chapter 5, *Things You Should Know about Analog Input*. SCXI channel, port, and counter addressing is described in Chapter 18, *Things You Should Know about SCXI*.

Onboard channels refer to analog or digital I/O channels provided by the plug-in DAQ device. If `x` is an onboard channel, you can specify this by entering `x` or `OBx` as the **channel list** element. Refer to the description of your device in your hardware user manual for restrictions on channel order. Figure 3-7 shows several ways you can address onboard channels `0`, `1`, and `2`. The top three examples apply to VIs whose channel parameters are string arrays. The bottom two examples apply to VIs whose channel parameters are scalar strings.

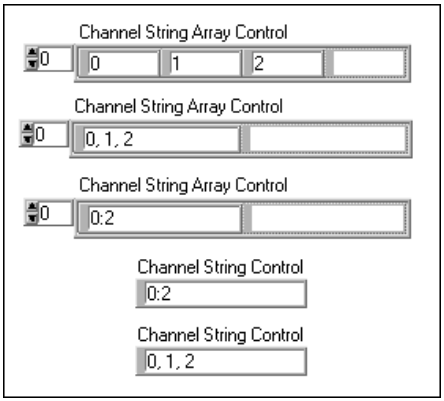



**Figure 3-7.** Channel String Array Controls

**Note** Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Function and VI Reference Manual* for the number of channels your device can acquire data from at one time, or refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**.

# Limit Settings

*Limit settings* are the maximum and minimum values of the analog signal(s) you are measuring or generating. The pair of limit setting values can be unique for each analog input or output channel. For analog input applications, the limit setting values must be within the range for the device. For more information on the range for your device, refer to Chapter 5, *Things You Should Know about Analog Input*.

Each pair of limit setting values forms a cluster. Analog output limits have a third member, the reference source. For simplicity, LabVIEW refers to limit settings as a pair of values. LabVIEW uses an array of these clusters to assign limits to the channels in your **channel** string array.

If you use the DAQ Channel Wizard to configure your analog input channels, the unit applied to the limit settings is the physical unit you specified for a particular channel name in the DAQ Channel Wizard. For example, if you configured a channel in the DAQ Channel Wizard to have physical units of `Deg C`, the limit settings are treated as limits in degrees Celsius. LabVIEW configures your hardware to make the measurement in terms of your channel name configuration. Unless you need to overwrite your channel name configuration, do not wire this input. Allow LabVIEW to set it up for you.

**Note** For NI-DAQ 6.5 and higher, the DAQ Channel Wizard is part of Measurement & Automation Explorer.

If you are not using the DAQ Channel Wizard, the default unit applied to the limit settings is usually volts, although the unit applied to the limit settings may be volts, current, resistance, or frequency, depending on the capability and configuration of your device.

The default range of the device, set in the configuration utility or by LabVIEW according to the channel name configuration in the DAQ Channel Wizard, is used whenever you leave the limit settings terminal unwired or you enter `0` for your upper and lower limits.

As the previous *Channel, Port, and Counter Addressing* section explains, LabVIEW uses an array of strings to specify which channels belong to a group. Also, remember LabVIEW lists one channel to as many as all of the device's channels in a single array element in the **channel** string array. LabVIEW also assigns all the channels listed in a **channel** string array element the same settings in the corresponding **limit settings** cluster array element. Figure 3-8 illustrates one case of this.

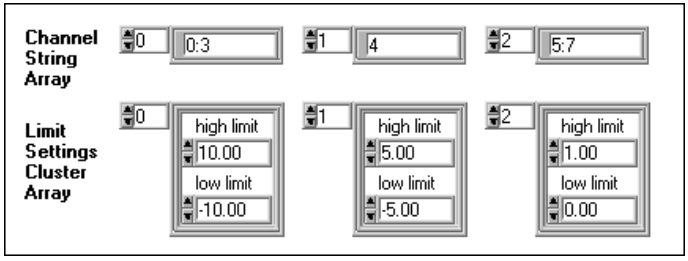



**Figure 3-8.** Limit Settings, Case 1

In this example, channels `0:3` (or 0, 1, 2, and 3) are assigned limits of `10.00` to –`10.00`. Channel 4 has limits of `5.00` to –`5.00`. Channels 5, 6, and 7 have limit settings of `1.00` to `0.00`.

If the **limit settings** cluster array has fewer elements than the **channel** string array, LabVIEW assigns any remaining channels the limit settings contained in the last entry of the **limit settings** cluster array. Figure 3-9 illustrates this case.

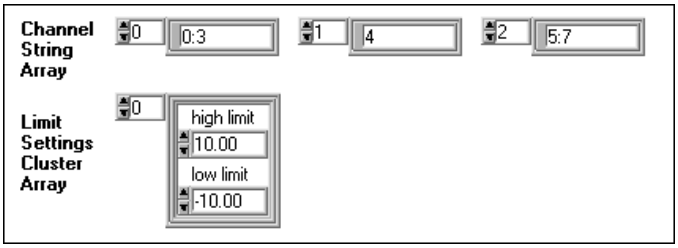



**Figure 3-9.** Limit Settings, Case 2

In this example, channels 0, 1, 2, and 3 have limits of `10.00` to `–10.00`. There are more channels left, but the **limit settings** cluster array is exhausted. Therefore, the remaining channels (4, 5, 6, and 7) are also assigned limits of `10.00` to `–10.00`.

The Easy Analog Input VIs have only one pair of input limits. This pair forms a single cluster element. If you specify the default limit settings, all channels scanned with these VIs will have identical limit settings. The Easy Analog Output VIs do not have limit settings. All the Intermediate VIs, both analog input and output, have the **channel** string array and the **limit settings** (or **input limits**) cluster array on the same VI. Assignment of limits to channels works exactly as described above. Refer to the *LabVIEW Function and VI Reference Manual* for more information on how to assign limit settings to a particular analog channel using the Advanced VIs, the Group Config VI and the Hardware Config VI, or refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**.

In analog applications, you not only specify the range of the signal, you must also specify the range and the polarity of the device. A *unipolar range* is a range containing either positive or negative values, but never both. A *bipolar range* is a range that has both positive and negative values. When a device uses jumpers or DIP switches to select its range and polarity, you must enter the correct jumper setting in the configuration utility.

In DAQ hardware manuals and in the configuration utility, you may find reference to the concept of *gain*. Gain is the amplification or attenuation of a signal. Most National Instruments DAQ devices have programmable gains (no jumpers), but some SCXI modules require the use of jumpers or DIP switches. For all DAQ devices used with LabVIEW, the gain is determined by limit settings. However, for some SCXI modules, you must enter the gain in the configuration utility.

# Data Organization for Analog Applications

If you acquire data from more than one channel multiple times, the data is returned as a two-dimensional (2D) array. If you were to create a 2D array and label the index selectors on a LabVIEW front panel, the array might look like Figure 3-10.

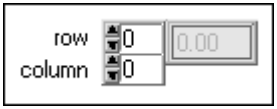



**Figure 3-10.** Example of a Basic 2D Array

The two vertically arranged boxes on the left are the row and column index selectors for the array. The top index selects a row, and the bottom index selects a column.

You can organize data for a 2D array in one of two ways. First, you can organize the data by rows. If the array contained data from analog input channels, this would mean that each row holds data from one channel. Selecting a row selects a channel. Selecting a column selects a scan of data. This ordering method is often referred to as *row major order*. When you create data in a nested For Loop, the inner loop creates a row for each iteration of the outer loop. If you were to label your index selectors for a row major 2D array, the array might look like Figure 3-11.

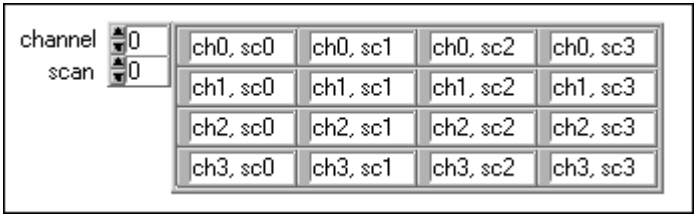



**Figure 3-11.** 2D Array in Row Major Order

You also can organize 2D array data by columns. The Analog Input VIs in LabVIEW organize their data in this way. Each column holds data from one channel, so selecting a column selects a channel. Selecting a row selects a scan of data. This ordering method is often called *column major order*. If you were to label your index selectors for a column major 2D array, the array might look like Figure 3-12.

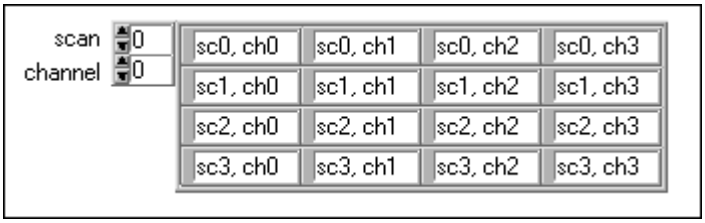



**Figure 3-12.** 2D Array in Column Major Order

To graph a column major order 2D array, you must configure the waveform chart or graph to treat the data as transposed by turning on this option in the graph pop-up menu.

**Note** This option appears in gray until you wire the 2D array to a graph. To convert the data to row major order, select **Functions»Array & Cluster»Transpose 2D Array**.

To extract a single channel from a column major 2D array, use the Index Array function from **Functions»Array & Cluster**. Add a dimension so that you have two black rectangles in the lower left corner. The top rectangle selects the row, and the bottom rectangle selects the column. Pop up on the row rectangle and select **Disable Indexing**. Now, when you select a column (or channel) by wiring your selection to the bottom rectangle, the Index Array function produces the entire column of data as a 1D array, as shown in Figure 3-13.

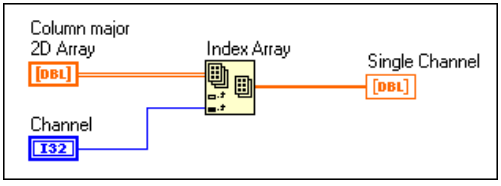



**Figure 3-13.** Extracting a Single Channel from a Column Major 2D Array

Analog output buffers that contain data for more than one channel are also column major 2D arrays. To create such an array, first make the data from each output channel a 1D array. Then select the Build Array function from **Functions»Array & Cluster**. Add as many input terminals (rows) to the Build Array terminal as you have channels of data. Wire each 1D array to the Build Array terminal to combine these arrays into a single row major 2D array. Then use the Transpose 2D Array function to convert the array to a column major array.

The finished array is ready for the AO Write VI, as shown in Figure 3-14.

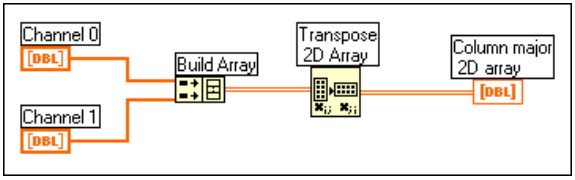



**Figure 3-14.** Analog Output Buffer 2D Array

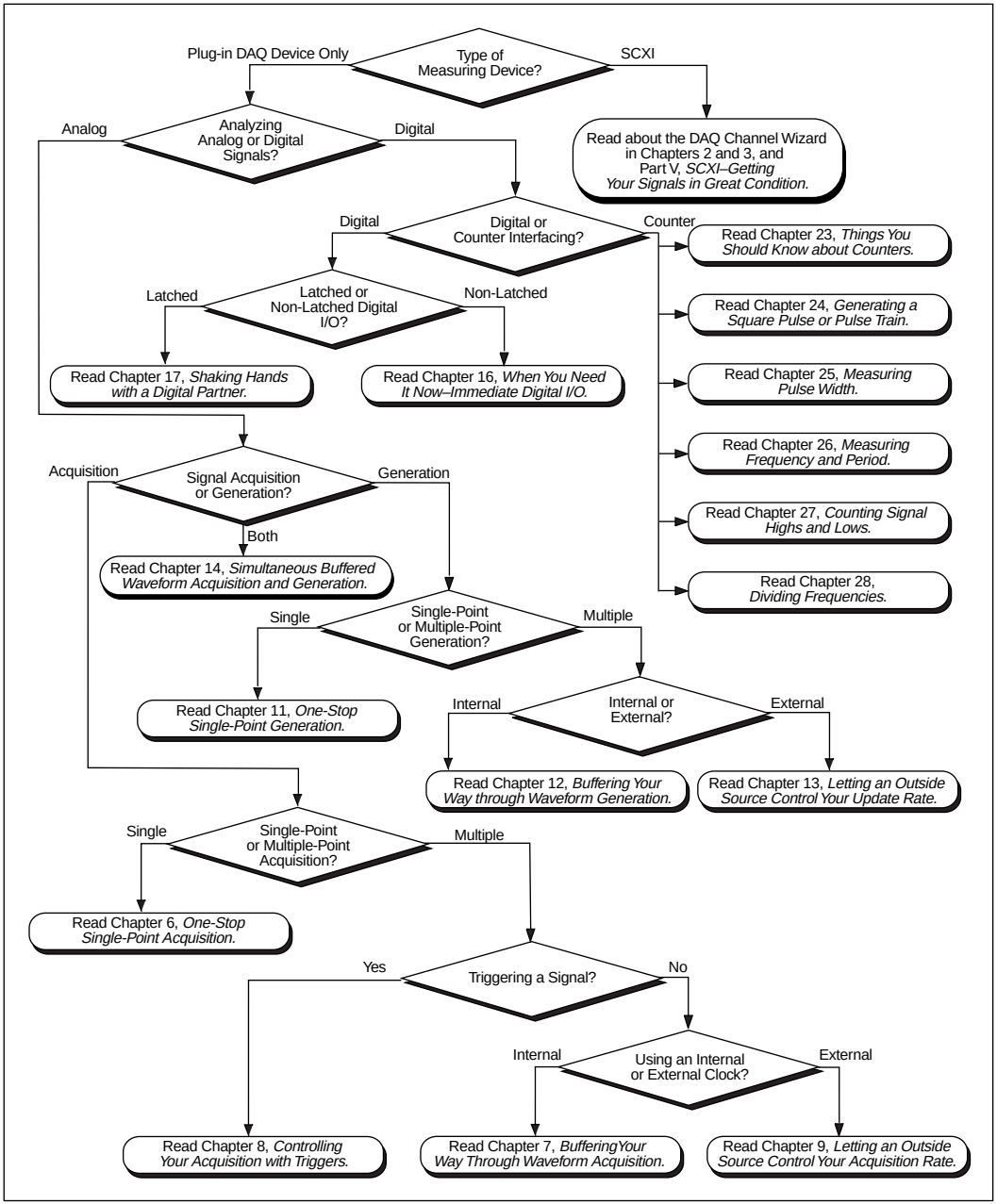
# 4

# Where You Should Go Next

This chapter directs you to the chapter in the manual best suited to answer questions about your data acquisition application. You answer a series of questions that help determine the purpose of your application. At first, the questions are general and then become more focused until you are referred to a specific section in the manual dealing with your type of application.

**Note** This manual is divided into parts. You always should read the *Things You Should Know About* chapter at the beginning of each part specific to your application. The *Things You Should Know About* chapters teach you about basic concepts dealing with your application.

Use the following flowchart as a guide as you answer the questions that follow it. The questions should pinpoint the sections in the manual you should read for your particular application.


Type of Measuring Device?
Plug-in DAQ Device Only
SCXI
Analyzing Analog or Digital Signals?
Analog
Digital
Read about the DAQ Channel Wizard in Chapters 2 and 3, and Part V, SCXI–Getting Your Signals in Great Condition.
Digital or Counter Interfacing?
Digital
Counter
Read Chapter 23, Things You Should Know about Counters.
Latched or Non-Latched Digital I/O?
Latched
Non-Latched
Read Chapter 24, Generating a Square Pulse or Pulse Train.
Read Chapter 17, Shaking Hands with a Digital Partner.
Read Chapter 16, When You Need It Now–Immediate Digital I/O.
Read Chapter 25, Measuring Pulse Width.
Read Chapter 26, Measuring Frequency and Period.
Signal Acquisition or Generation?
Acquisition
Generation
Read Chapter 27, Counting Signal Highs and Lows.
Both
Read Chapter 14, Simultaneous Buffered Waveform Acquisition and Generation.
Read Chapter 28, Dividing Frequencies.
Single-Point or Multiple-Point Generation?
Single
Multiple
Read Chapter 11, One-Stop Single-Point Generation.
Internal or External?
Internal
External
Read Chapter 12, Buffering Your Way through Waveform Generation.
Read Chapter 13, Letting an Outside Source Control Your Update Rate.
Single-Point or Multiple-Point Acquisition?
Single
Multiple
Read Chapter 6, One-Stop Single-Point Acquisition.
Triggering a Signal?
Yes
No
Using an Internal or External Clock?
Internal
External
Read Chapter 8, Controlling Your Acquisition with Triggers.
Read Chapter 7, BufferingYour Way Through Waveform Acquisition.
Read Chapter 9, Letting an Outside Source Control Your Acquisition Rate.

# Questions You Should Answer

1. **Measuring Device—DAQ Device or SCXI Module?**

   Are you working in a noisy environment? If you are, then you may have SCXI modules connected to your DAQ device or the parallel port of your computer. SCXI modules can filter and isolate noise from signals. They can also amplify low signals. SCXI modules expand the number of channels to acquire or generate data.

   DAQ devices are primarily used alone when extra signal conditioning is not necessary.

   If you are using a DAQ device, then read question 2. If you are using SCXI, go to Part V, *SCXI—Getting Your Signals in Great Condition*.

2. **Analog or Digital Signal Analysis?**

   Does your signal have two discrete values that are TTL signals? If so, then you have a digital signal. Otherwise, you have an analog signal. The type of information you would need to know from an analog signal is the level (discrete value), shape, and frequency content.

3. **Analog or Digital Signal Acquisition or Generation?**

   If you want to measure and analyze signals from a source outside the computer, you want to *acquire* signals. If you want to send signals to an outside instrument to control its operation, then you want to *generate* signals.

   If you want to *acquire* analog signals, go to question 5. If you want to *generate* analog signals, refer to question 6. If you want to acquire and generate analog signals, refer to the *Using Analog Input/Output Control Loops* section of Chapter 6, *One-Stop Single-Point Acquisition*.

   If you want to acquire or generate *digital* signals, read the next question.

**4. Digital or Counter Interfacing?**

Digital I/O interfaces primarily with binary operations, such as turning external equipment on or off or sense logic states, such as the on/off position of the switch. Counters generate individual digital pulses or waves or count digital events, like how many times a digital signal rises or falls in value.

If you are performing digital I/O, refer to question 9. If you need to use counters, read question 10.

**5. Single-Point or Multiple-Point Acquisition?**

Do you want to acquire a signal value(s) at one time or over a period of time at a certain rate? If you measure a signal at a given instant of time, you are performing *single-point acquisition*. If you measure signals over a period of time at a certain rate, then you are performing *multiple-point* or *waveform acquisition.*

If you want single-point acquisition, refer to Chapter 6, *One-Stop Single-Point Acquisition*. If you want multiple-point acquisition, read question 7.

**6. Single-Point or Multiple-Point Generation?**

Are you outputting a steady (DC) signal or are you generating a changing signal at a certain rate? A constant or slowly changing signal output is called *single-point generation*. The output of a changing signal at a certain rate is called *multiple-point* or *waveform generation*.

If you want to perform single-point generation, refer to Chapter 11, *One-Stop Single-Point Generation*. If you want multiple-point generation, refer to Chapter 12, *Buffering Your Way through Waveform Generation*.

**7. Triggering a Signal or Using a Clock?**

You can start an analog acquisition when a certain analog or digital value occurs by *triggering* the acquisition.

If you want to trigger an analog acquisition, refer to Chapter 8, *Controlling Your Acquisition with Triggers*.

**8. Multiple-Point Acquisition with an Internal or External Clock?**

Multiple point or waveform acquisition can be done at a rate set by an internal DAQ device clock or an external clock. The external clock will be a TTL signal produced at a certain rate.

If you want to acquire a waveform at the rate of an external signal, refer to Chapter 9, *Letting an Outside Source Control Your Acquisition Rate*. If not, read Chapter 7, *Buffering Your Way through Waveform Acquisition*.

**9. Non-Latched or Latched Digital I/O?**

If you want your program to read the latest digital input or immediately write a new digital output value, you should use non-latched (immediate) digital I/O. When a DAQ device accepts or transfers data after a digital pulse has been received, it is called latched (handshaked) digital I/O. With latched digital I/O, you can store the values you want to transfer in a buffer. Only one value will be transferred after each handshaked pulse.

If you want to use non-latched (immediate) digital I/O, refer to Chapter 16, *When You Need It Now— Immediate Digital I/O*. If you need to perform latched (handshaked) digital I/O, refer to Chapter 17, *Shaking Hands with a Digital Partner*.

**10. Counters—Counting or Generating Digital Pulses?**

If you want to generate digital pulses from a counter at a certain rate, read Chapter 24, *Generating a Square Pulse or Pulse Trains*. If you want to measure the width of a digital pulse, refer to Chapter 25, *Measuring Pulse Width*. If you want to measure the frequency or period of a digital signal, refer to Chapter 26, *Measuring Frequency and Period*. If you just want to count how many times a digital signal rises or falls, refer to Chapter 27, *Counting Signal Highs and Lows*. To learn how to slow the frequency of a digital signal, refer to Chapter 28, *Dividing Frequencies*.

# Part II

# Catching the Wave with Analog Input

This section contains basic information about acquiring data with LabVIEW, including acquiring a single point or multiple points, triggering your acquisition, and using outside sources to control acquisition rates.

Part II, *Catching the Wave with Analog Input*, contains the following chapters:

- Chapter 5, *Things You Should Know about Analog Input*, explains basic concepts on using analog input with LabVIEW.
- Chapter 6, *One-Stop Single-Point Acquisition*, shows you how to acquire one data point from a single channel and then one data point from each of several channels using LabVIEW, and explains how software-timing and/or hardware-timing affects the performance of analog I/O.
- Chapter 7, *Buffering Your Way through Waveform Acquisition*, reviews the different methods of reading multiple channels and explains how LabVIEW stores the acquired data with each method.
- Chapter 8, *Controlling Your Acquisition with Triggers*, explains how to set your analog acquisition to occur at a certain time using either software or hardware triggering methods.
- Chapter 9, *Letting an Outside Source Control Your Acquisition Rate*, shows you how to control your data acquisition rate by some other external source in your system.

# 5

# Things You Should Know about Analog Input

Hunting has been a part of survival from the beginning of time. People used to hunt for the things they needed to survive, like food and water. Today, engineers and scientists use data acquisition to "hunt down" the information they need to survive in the information age. This chapter focuses on defining the tools you need to be a successful hunter in the world of data acquisition.

## Defining Your Signal

You and your friends plan a hunting trip for this weekend. What do you plan to bring with you? This question is really not valid, because you must know first *what* you will be hunting before you pack your fishing pole or elephant rifle. The same idea applies to scientists and engineers engaged in the quest for information. You must know the defining characteristics of what you want to "hunt," be it a wild animal or an analog signal. You cannot just say, "I will hunt voltages," or even "I will hunt analog voltages." Voltages come in various forms. This chapter gives you the terms, tools, and techniques designed to help show you the best way to catch your wave.

Analog signals can be divided into three categories: DC, time domain, and frequency domain. You must ask yourself, "Is the information I seek primarily contained in the level, the shape, or the frequency content of my signal?" Figure 5-1 illustrates which signals correspond to certain types of signal information.

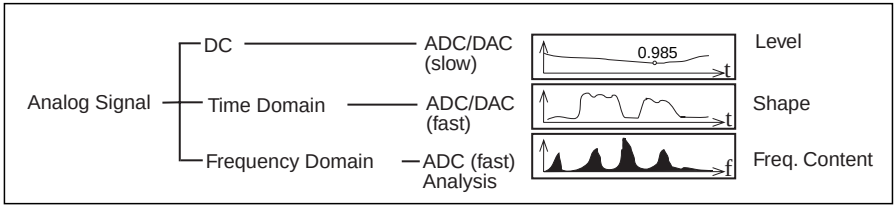



**Figure 5-1.** Types of Analog Signals

You might be saying to yourself, “I know that I have a thermocouple and that the primary information (temperature) is contained in the level of the analog voltage. Now I am ready to go hunting!” Well, you are *almost* ready to hunt, but you first must figure out a few more signal characteristics before you can begin. For example, to what is your signal referenced? How fast does the signal vary with time? The rate you sample determines how often the A/D conversions take place. A fast sampling rate acquires more points in a given time, and therefore can often form a better representation of the original signal than a slow sampling rate. According to the Nyquist Theorem, you must sample at a rate greater than twice the maximum frequency component in that signal to get accurate frequency information about that signal. The frequency at one half the sampling frequency is referred to as the *Nyquist frequency.* For more information on the Nyquist frequency, refer to the section *Sampling Considerations* in Chapter 11, *Introduction to Analysis in LabVIEW,* of the *LabVIEW User Manual*.

## What Is Your Signal Referenced To?

Signals come in two forms: *referenced* and *non-referenced* signal sources. More often, referenced sources are said to be *grounded* signals, and non-referenced sources are called *floating* signals.

### Grounded Signal Sources

Grounded signal sources have voltage signals that are referenced to a system ground, such as earth or a building ground. Devices that plug into a building ground through wall outlets, such as signal generators and power supplies, are the most common examples of grounded signal sources, as shown in Figure 5-2.

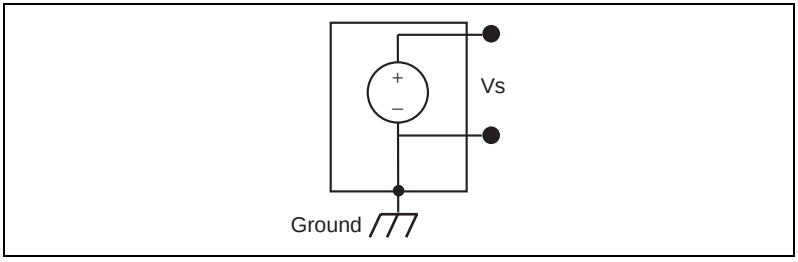



**Figure 5-2.** Grounded Signal Sources

## Floating Signal Sources

Floating signal sources contain a signal, such as a voltage, that is not connected to an absolute reference, such as earth or a building ground. Some common examples of floating signals are batteries, battery-powered sources, thermocouples, transformers, isolation amplifiers, and any instrument that explicitly floats its output signal. Notice that in Figure 5-3 neither terminal of the floating source is connected to the electrical outlet ground.

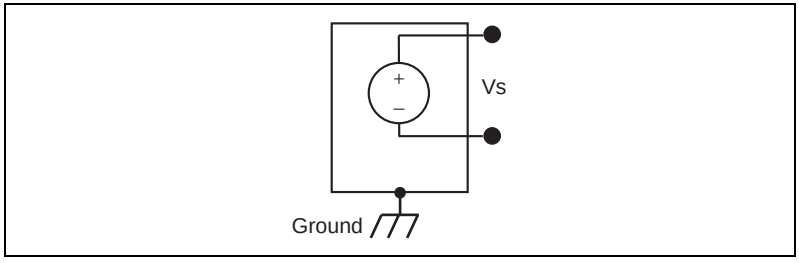



**Figure 5-3.** Floating Signal Sources

Now that you know how your signal is referenced, read on to learn about the different systems available to acquire these signals.

# Choosing Your Measurement System

Now that you have defined your signal, you must choose a measurement system. You have an analog signal, so you must convert the signal with an ADC measurement system, which converts your signal into information the computer can understand. Some of the issues you must resolve before choosing a measurement system are your ADC bit resolution, device range, and signal range.

## Resolution

The number of bits used to represent an analog signal determines the *resolution* of the ADC. You can compare the resolution on a DAQ device to the marks on a ruler. The more marks you have, the more precise your measurements. Similarly, the higher the resolution, the higher the number of divisions into which your system can break down the ADC range, and therefore, the smaller the detectable change. A 3-bit ADC divides the range into $2^3$ or 8 divisions. A binary or digital code between 000 and 111 represents each division. The ADC translates each measurement of the analog signal to one of the digital divisions. Figure 5-4 shows a sine wave digital image as obtained by a 3-bit ADC. Clearly, the digital signal does not represent the original signal adequately, because the converter has too few digital divisions to represent the varying voltages of the analog signal. By increasing the resolution to 16 bits, however, the ADC's number of divisions increases from 8 to 65,536 ($2^{16}$). The ADC can now obtain an extremely accurate representation of the analog signal.

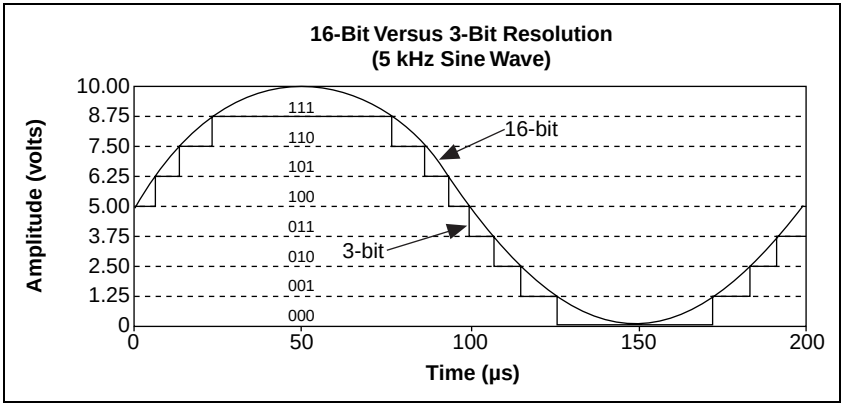



**Figure 5-4.** The Effects of Resolution on ADC Precision

## Device Range

Range refers to the minimum and maximum analog signal levels that the ADC can digitize. Many DAQ devices feature selectable ranges, so you can match the ADC range to that of the signal to take best advantage of the available resolution. For example, in Figure 5-5, the 3-bit ADC, as shown in the left chart, has eight digital divisions in the range from 0 to 10 V. If you select a range of –`10.00` to `10.00` V, as shown in the right chart, the same ADC now separates a 20 V range into eight divisions. The smallest detectable voltage increases from `1.25` to `2.50` V, and you now have a much less accurate representation of the signal.

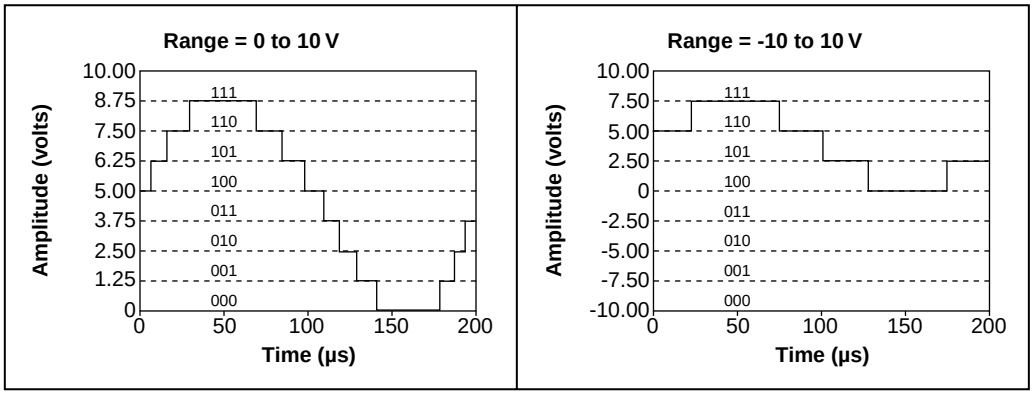



**Figure 5-5.** The Effects of Range on ADC Precision

## Signal Limit Settings

Limit settings are the maximum and minimum values of the signal you are measuring. A more precise limit setting allows the ADC to use more digital divisions to represent the signal. Figure 5-6 shows an example of this theory. Using a 3-bit ADC and a device range setting of `0.00` to `10.00` V, Figure 5-6 shows the effects of a limit setting between 0 and 5 V and 0 and 10 V. With a limit setting of 0 to 10 V, the ADC uses only four of the eight divisions in the conversion. But using a limit setting of 0 to 5 V, the ADC now has access to all eight digital divisions. This makes the digital representation of the signal more accurate.

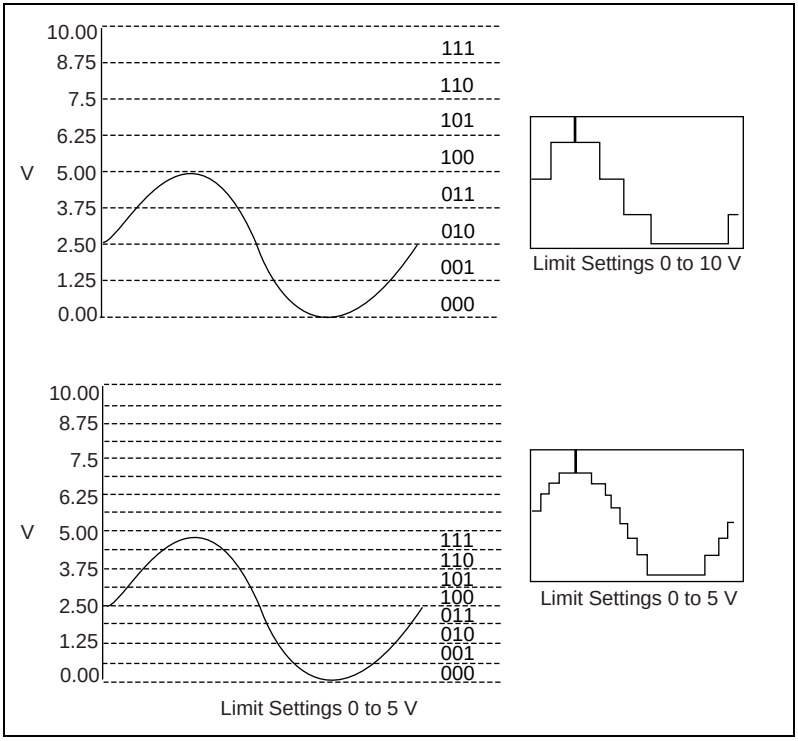



**Figure 5-6.** The Effects of Limit Settings on ADC Precision

# Considerations for Selecting Analog Input Settings

The resolution and device range of a DAQ device determine the smallest detectable change in the input signal. You can calculate the smallest detectable change, called the *code width*, using the following formula.

$$code\ width = \frac{device\ range}{2^{resolution}}$$

For example, a 12-bit DAQ device with a 0 to 10 V input range detects a 2.4 mV change, while the same device with a –10 to 10 V input range detects only a change of 4.8 mV.

$$\frac{device\ range}{2^{resolution}} = \frac{10}{2^{12}} = 2.4\ mV$$

$$\frac{device\ range}{2^{resolution}} = \frac{20}{2^{12}} = 4.8\ mV$$

A high-resolution A/D converter provides a smaller code width given the device voltage ranges shown above.

$$\frac{device\ range}{2^{resolution}} = \frac{10}{2^{16}} = .15\ mV$$

$$\frac{device\ range}{2^{resolution}} = \frac{20}{2^{16}} = .3\ mV$$

The smaller your code width, the more accurate your measurements will be.

There are times you must know whether your signals are unipolar or bipolar. Unipolar signals are signals that range from 0 value to a positive value (for example, 0 to 5 V). Bipolar signals are signals that range from a negative to a positive value (for example, –5 to 5 V). To achieve a smaller code width if your signal is unipolar, specify that the device range is unipolar, as shown previously. If your signal range is smaller than the device range, you should set your limit settings to values that more accurately reflect your signal range. Table 5-1 shows how the code width of the 12-bit DAQ devices vary with device ranges and limit settings, because your limit settings automatically adjust the gain on your device.

**Table 5-1.** Measurement Precision for Various Device Ranges and Limit Settings (12-bit A/D Converter)

| Device Voltage Range | Limit Settings | Precision[1] |
|---|---|---|
| 0 to 10 V | 0 to 10 V | 2.44 mV |
| | 0 to 5 V | 1.22 mV |
| | 0 to 2.5 V | 610 µV |
| | 0 to 1.25 V | 305 µV |
| | 0 to 1 V | 244 µV |
| | 0 to 0.1 V | 24.4 µV |
| | 0 to 20 mV | 4.88 µV |
| –5 to 5 V | –5 to 5 V | 2.44 mV |
| | –2.5 to 2.5 V | 1.22 mV |
| | –1.25 to 1.25 V | 610 µV |
| | –0.625 to 0.625 V | 305 µV |
| | –0.5 to 0.5 V | 244 µV |
| | –50 to 50 mV | 24.4 µV |
| | –10 to 10 mV | 4.88 µV |
| –10 to 10 V | –10 to 10 V | 4.88 mV |
| | –5 to 5 V | 2.44 mV |
| | –2.5 to 2.5 V | 1.22 mV |
| | –1.25 to 1.25 V | 610 µV |
| | –1 to 1 V | 488 µV |
| | –0.1 to 0.1 V | 48.8 µV |
| | –20 to 20 mV | 9.76 µV |

[1] The value of 1 Least Significant Bit (LSB) of the 12-bit ADC. In other words, the voltage increment corresponding to a change of 1 count in the ADC 12-bit count.

For more information on the device range and limit settings for your device, refer to the tables in Appendix B, *Hardware Capabilities*, in the *LabVIEW Function and VI Reference Manual*, or to the LabVIEW *Online Reference*, available by selecting **Help»Online Reference**. In these tables, there is information on gain settings for each device. For more information on gain, refer to the *Limit Settings* section of Chapter 3, *Basic LabVIEW Data Acquisition Concepts*.

Now that you know which kind of ADC to use and what settings to use for your signal, you can connect your signals to be measured. On most DAQ devices, there are three different ways to configure your device to read the signals: differential, referenced single-ended (RSE), and nonreferenced single-ended (NRSE).

## Differential Measurement System

In a differential measurement system, you do not need to connect either input to a fixed reference, such as earth or a building ground. DAQ devices with instrumentation amplifiers can be configured as differential measurement systems. Figure 5-7 depicts the 8-channel differential measurement system used in the MIO series devices. Analog multiplexers increase the number of measurement channels while still using a single instrumentation amplifier. For this device, the pin labeled AIGND (the analog input ground) is the measurement system ground.

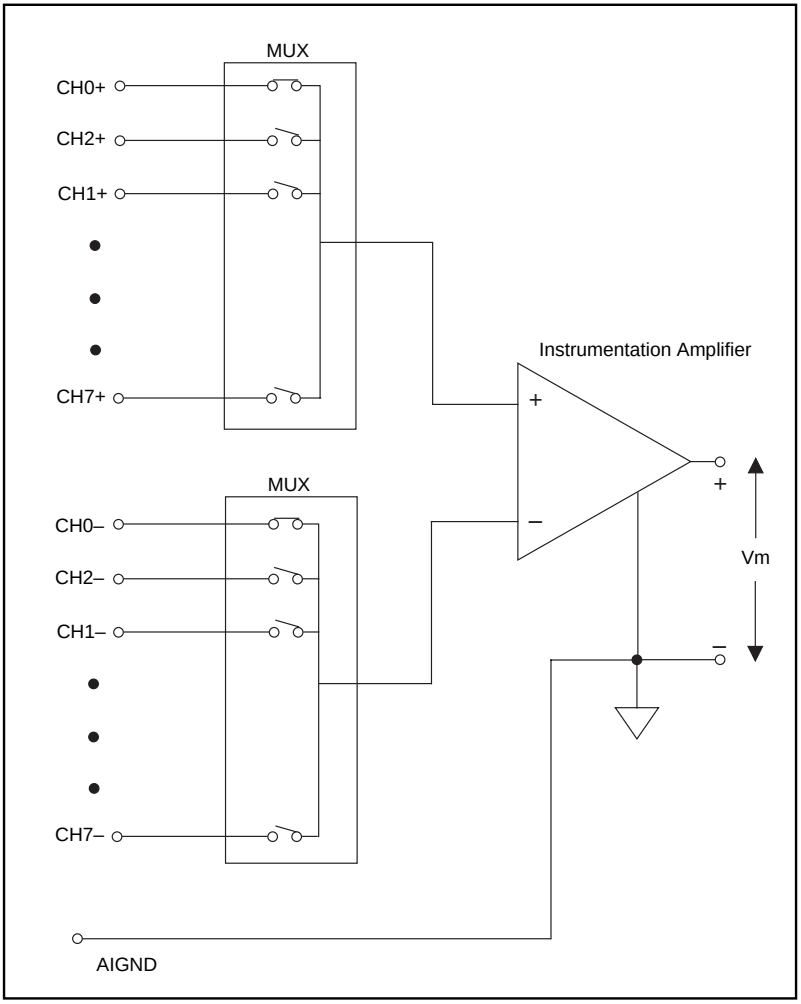



**Figure 5-7.** 8-Channel Differential Measurement System

In general, a differential measurement system is preferable because it rejects not only ground loop-induced errors, but also the noise picked up in the environment to a certain degree. Use differential measurement systems when all input signals meet the following criteria:

- Low-level signals (for example, less than 1 V)
- Long or non-shielded cabling/wiring traveling through a noisy environment
- Any of the input signals require a separate ground-reference point or return signal

An ideal differential measurement system reads only the potential difference between its two terminals—the positive (+) and negative (–) inputs. Any voltage present at the instrumentation amplifier inputs with respect to the amplifier ground is called a *common-mode voltage*. An ideal differential measurement system completely rejects (does not measure) common-mode voltage, as shown in Figure 5-8.

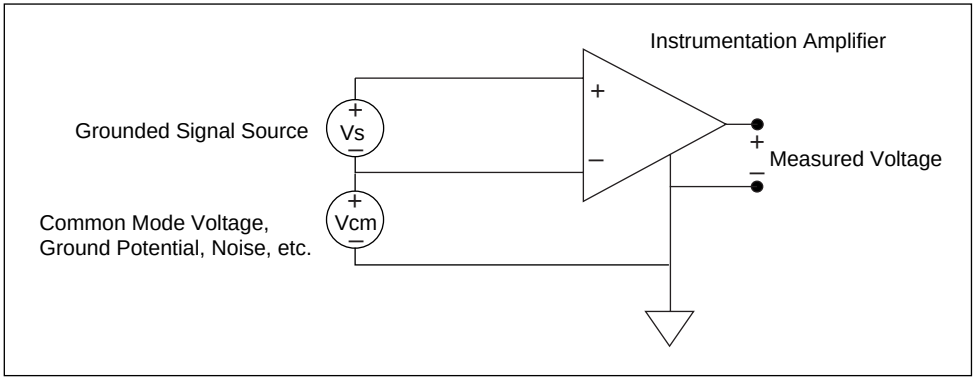



**Figure 5-8.** Common-Mode Voltage

While a differential measurement system is often the best choice, a single-ended configuration uses twice as many measurement channels. A single-ended measurement system is acceptable when the magnitude of the induced errors is smaller than the required accuracy of the data.

## Referenced Single-Ended Measurement System

An RSE measurement system is used to measure a floating signal, because it grounds the signal with respect to building ground. Figure 5-9 depicts a 16-channel RSE measurement system. You only should use this measurement system when you need a single-ended system, and your device does not work with NRSE measurement.

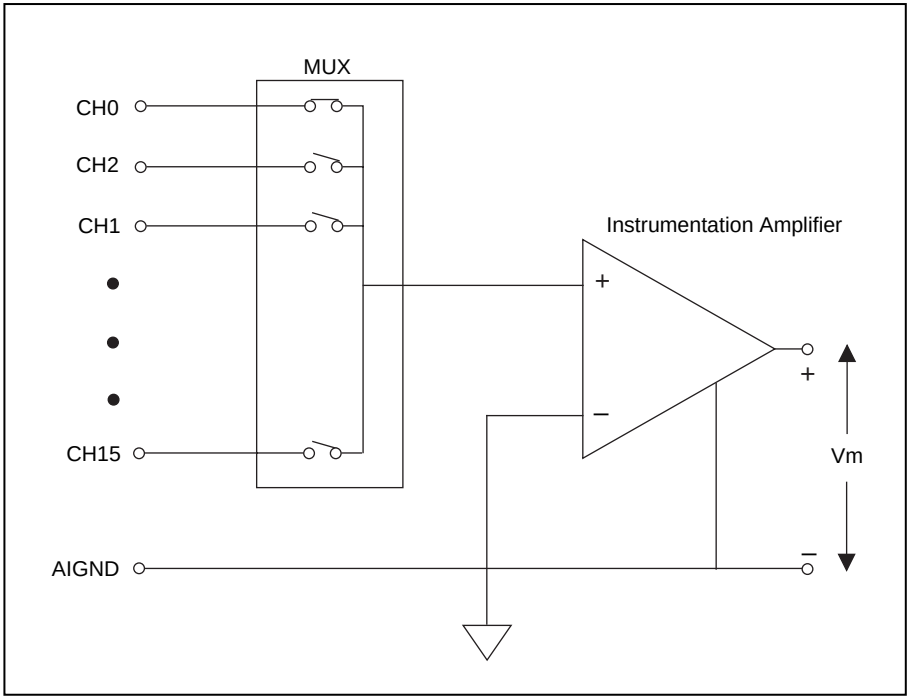



**Figure 5-9.** 16-Channel RSE Measurement System

## Nonreferenced Single-Ended Measurement System

DAQ devices often use a variant of the RSE measurement technique, known as the NRSE measurement system. In an NRSE measurement system, all measurements are made with respect to a common reference, because all of the input signals are already grounded. Figure 5-10 depicts an NRSE measurement system where AISENSE is the common reference for taking measurements and AIGND is the system ground. All signals must share a common reference at AISENSE.

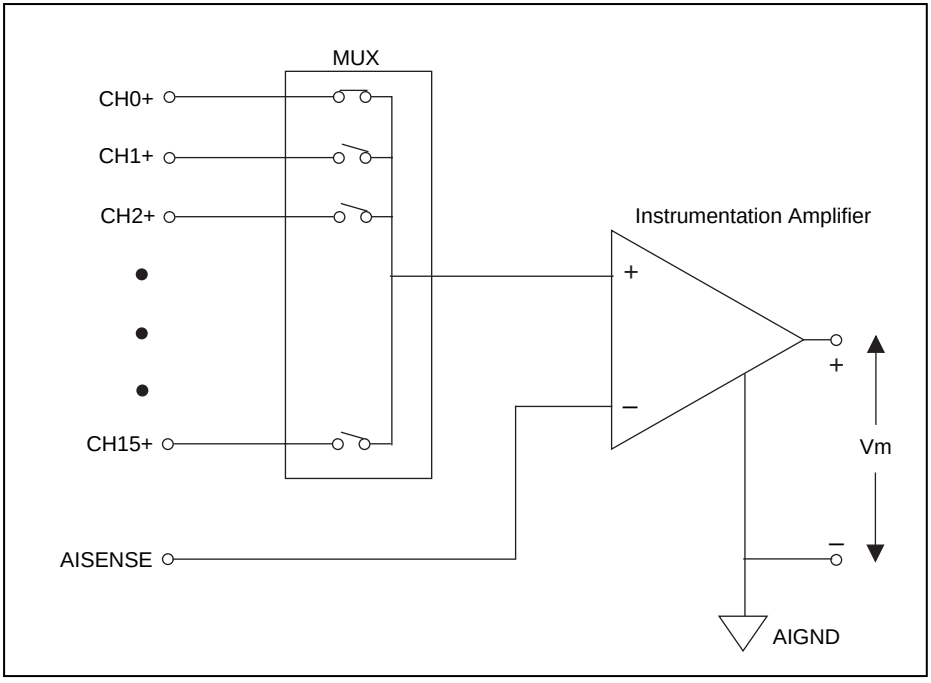



**Figure 5-10.** 16-Channel NRSE Measurement System

In general, a differential measurement system is preferable because it rejects not only ground loop-induced errors, but also the noise picked up in the environment to a certain degree. On the other hand, the single-ended configuration allows for twice as many measurement channels and is acceptable when the magnitude of the induced errors is smaller than the required accuracy of the data.

You can use single-ended measurement systems when all input signals meet the following criteria:

- High-level signals (normally, greater than 1 V)
- Short or properly-shielded cabling/wiring traveling through a noise-free environment (normally, less than 15 ft)
- All signals can share a common reference signal at the source

Use differential connections when your system violates any of the above criteria.

# Channel Addressing with the AMUX-64T

An AMUX-64T external multiplexer device expands the number of analog input signals a plug-in DAQ device can measure. You can address AMUX-64T channels when you attach one, two, or four AMUX-64T devices to a plug-in DAQ device. With this device, you can multiplex four, eight, or 16 AMUX-64T channels into one device channel. The scanning order of these AMUX-64T channels is fixed. To specify a range of AMUX-64T channels, enter the device channel into which the range is multiplexed in the **channel list**. For example, if you have no AMUX-64T devices, a **channel list** element of `0` specifies device channel `0`. If you have an AMUX-64T device, a **channel list** element of `0` specifies channels `0` through `3` on each AMUX-64T device. Table 5-2 shows the number of channels available on a DAQ device with an external multiplexer.

**Table 5-2.** Analog Input Channel Range

| Number of AMUX-64Ts | Channel Range (Single-Ended) | Channel Range (Differential) |
|---|---|---|
| 0 | 0 through 15 | 0 through 7 |
| 1 | `AM1!0` through `AM1!63` | `AM1!0` through `AM1!31` |
| 2 | `AM1!0` through `AM1!63`, `AM2!0` through `AM2!63` | `AM1!0` through `AM1!31`, `AM2!0` through `AM2!31` |
| 4 | `AM1!0` through `AM1!63`, `AM2!0` through `AM2!63`, `AM3!0` through `AM3!63`, `AM4!0` through `AM4!63` | `AM1!0` through `AM1!31`, `AM2!0` through `AM2!31`, `AM3!0` through `AM3!31`, `AM4!0` through `AM4!31` |

You specify the number of AMUX devices through the configuration utility or the AI Hardware Config VI. Refer to the *LabVIEW Function and VI Reference Manual* or the LabVIEW *Online Reference*, available by selecting **Help»Online Reference**, for more information on this VI.

## The AMUX-64T Scanning Order

This section explains how LabVIEW scans channels from the AMUX-64T. You must know this scanning order so that you can determine which analog input channel LabVIEW scanned during a data acquisition operation.

The scanning counters on the AMUX-64T and on the DAQ device perform automatic scanning of the AMUX-64T analog input channels. When you perform a multiple-channel scanned data acquisition with an AMUX-64T, a counter on the DAQ device switches the DAQ device multiplexers.

When you connect a single AMUX-64T device to the DAQ device, you must scan four AMUX-64T input channels for every DAQ device channel. If you attach two AMUX-64T devices to the DAQ device, LabVIEW scans eight AMUX-64T channels for every DAQ device input channel. For example, assume that channels 0 through 3 on AMUX-64T device 1 and channels 0 through 3 on AMUX-64T device 2 are multiplexed together into DAQ device channel 0. In this case, LabVIEW scans the first four channels on AMUX-64T device 1, followed by the first four channels on AMUX-64T device 2.

If you attach four AMUX-64T devices to the DAQ device, LabVIEW scans 16 AMUX-64T channels for every DAQ device input channel. For example, channels 0 through 3 on AMUX-64T device 1, 2, 3, and 4 are multiplexed together into DAQ device channel 0. In this case, LabVIEW scans the first four channels on device 1, followed by the first four channels on device 2, the first four channels on device 3, and then the first four channels on device 4.

The order in which LabVIEW scans channels depends on the channel list you specify in the AI Group Config VI. You specify this channel list as an array of DAQ device channel numbers indicating the order in which LabVIEW scans the DAQ device channels. When scanning multiple channels, list only the device channels—not the AMUX-64T channels. (You only use the `AMy!x` syntax in your channel list when you sample a single AMUX-64T channel.) LabVIEW then scans four, eight, or 16 channels for every device channel for one, two, or four AMUX-64T devices, respectively. However, the AMUX-64T has a fixed scanning order. Table 5-3 shows the order in which LabVIEW scans the AMUX-64T channels for every DAQ device input channel when you use one or two AMUX-64T devices. Table 5-4 shows the order in which LabVIEW scans the AMUX-64T channels for every DAQ device input channel when you use four AMUX-64T devices.

If you want to scan more than one AMUX-64T channel, you must enter the device channels in your scan list.

**Table 5-3.** Scanning Order for Each DAQ Device Input Channel with One or Two AMUX-64Ts

| DAQ Device Channel | AMUX-64T Channels | | |
|---|---|---|---|
| | One Device | Two Devices | |
| | Device 1 | Device 1 | Device 2 |
| 0 | 0, 1, 2, 3 | 0, 1, 2, 3 | 0, 1, 2, 3 |
| 1 | 4, 5, 6, 7 | 4, 5, 6, 7 | 4, 5, 6, 7 |
| 2 | 8, 9, 10, 11 | 8, 9, 10, 11 | 8, 9, 10, 11 |
| 3 | 12, 13, 14, 15 | 12, 13, 14, 15 | 12, 13, 14, 15 |
| 4 | 16, 17, 18, 19 | 16, 17, 18, 19 | 16, 17, 18, 19 |
| 5 | 20, 21, 22, 23 | 20, 21, 22, 23 | 20, 21, 22, 23 |
| 6 | 24, 25, 26, 27 | 24, 25, 26, 27 | 24, 25, 26, 27 |
| 7 | 28, 29, 30, 31 | 28, 29, 30, 31 | 28, 29, 30, 31 |
| 8 | 32, 33, 34, 35 | 32, 33, 34, 35 | 32, 33, 34, 35 |
| 9 | 36, 37, 38, 39 | 36, 37, 38, 39 | 36, 37, 38, 39 |
| 10 | 40, 41, 42, 43 | 40, 41, 42, 43 | 40, 41, 42, 43 |
| 11 | 44, 45, 46, 47 | 44, 45, 46, 47 | 44, 45, 46, 47 |
| 12 | 48, 49, 50, 51 | 48, 49, 50, 51 | 48, 49, 50, 51 |
| 13 | 52, 53, 54, 55 | 52, 53, 54, 55 | 52, 53, 54, 55 |
| 14 | 56, 57, 58, 59 | 56, 57, 58, 59 | 56, 57, 58, 59 |
| 15 | 60, 61, 62, 63 | 60, 61, 62, 63 | 60, 61, 62, 63 |

**Table 5-4.** Scanning Order for Each DAQ Device Input Channel with Four AMUX-64Ts

| DAQ Device Channel | AMUX-64T Channels | | | |
|---|---|---|---|---|
| | Device 1 | Device 2 | Device 3 | Device 4 |
| 0 | 0, 1, 2, 3 | 0, 1, 2, 3 | 0, 1, 2, 3 | 0, 1, 2, 3 |
| 1 | 4, 5, 6, 7 | 4, 5, 6, 7 | 4, 5, 6, 7 | 4, 5, 6, 7 |
| 2 | 8, 9, 10, 11 | 8, 9, 10, 11 | 8, 9, 10, 11 | 8, 9, 10, 11 |
| 3 | 12, 13, 14, 15 | 12, 13, 14, 15 | 12, 13, 14, 15 | 12, 13, 14, 15 |
| 4 | 16, 17, 18, 19 | 16, 17, 18, 19 | 16, 17, 18, 19 | 16, 17, 18, 19 |
| 5 | 20, 21, 22, 23 | 20, 21, 22, 23 | 20, 21, 22, 23 | 20, 21, 22, 23 |
| 6 | 24, 25, 26, 27 | 24, 25, 26, 27 | 24, 25, 26, 27 | 24, 25, 26, 27 |
| 7 | 28, 29, 30, 31 | 28, 29, 30, 31 | 28, 29, 30, 31 | 28, 29, 30, 31 |
| 8 | 32, 33, 34, 35 | 32, 33, 34, 35 | 32, 33, 34, 35 | 32, 33, 34, 35 |
| 9 | 36, 37, 38, 39 | 36, 37, 38, 39 | 36, 37, 38, 39 | 36, 37, 38, 39 |
| 10 | 40, 41, 42, 43 | 40, 41, 42, 43 | 40, 41, 42, 43 | 40, 41, 42, 43 |
| 11 | 44, 45, 46, 47 | 44, 45, 46, 47 | 44, 45, 46, 47 | 44, 45, 46, 47 |
| 12 | 48, 49, 50, 51 | 48, 49, 50, 51 | 48, 49, 50, 51 | 48, 49, 50, 51 |
| 13 | 52, 53, 54, 55 | 52, 53, 54, 55 | 52, 53, 54, 55 | 52, 53, 54, 55 |
| 14 | 56, 57, 58, 59 | 56, 57, 58, 59 | 56, 57, 58, 59 | 56, 57, 58, 59 |
| 15 | 60, 61, 62, 63 | 60, 61, 62, 63 | 60, 61, 62, 63 | 60, 61, 62, 63 |

To determine which AMUX-64T channels LabVIEW scans and the scanning order, perform the following steps:

1. Locate the channel for each DAQ device channel in your channel list in the DAQ Device Channel column in Tables 5-3 or 5-4. Start with the first device channel and continue through the list in your specified channel order.
2. Read from left to right along the table row where you located the channel number to find the AMUX-64T scanning order.

To read a single AMUX-64T channel, use channel specifier `AMy!x`. This specifier returns data from channel `x` of the AMUX-64T with ID `y`. To read more than one AMUX-64T channel, use channel specifier `OBx:y`. This specifier returns data from the AMUX-64T channels that correspond to device channel `x` through device channel `y`.

When the **channel list** contains a single AMUX-64T channel, you must also specify the number of the AMUX-64T device, as shown in the following table.

| Channel List Parameter | Channel Specified |
|---|---|
| AMy!x | Channel x on AMUX-64T device y. |
| AM4!8 | Channel 8 on AMUX-64T device 4. |

You refer to AMUX-64T channels only when a single AMUX-64T channel comprises the entire list. Otherwise, you refer to them indirectly through the device channels that you use to scan the AMUX-64T channels. Refer to Appendix B, *Hardware Capabilities*, of the *LabVIEW Function and VI Reference Manual*, or the LabVIEW *Online Reference*, available by selecting **Help»Online Reference**, for more information on addressing AMUX-64T channels.

Refer to the *AMUX-64T User Manual* for more information on the external multiplexer device.

# Important Terms You Should Know

The following are some definitions of common terms and parameters that you should remember when acquiring your data:

- A *scan* is one acquisition or reading from each channel in your channel string.
- **Number of scans to acquire** refers to the number of data acquisitions or readings to acquire from each channel in the channel string. **Number of samples** is the number of data points you want to sample from each channel.
- The **scan rate** determines how many times per second LabVIEW acquires data from channels. **scan rate** enables *interval scanning* (a longer interval between scans than between individual channels comprising a scan) on devices that support this feature. **channel clock rate** defines the time between the acquisition of consecutive channels in your channel string. For more information on scan and channel clock rates, refer to Chapter 9, *Letting an Outside Source Control Your Acquisition Rate*.

For specific information about the Analog Input VIs, refer to Chapter 14, *Introduction to the LabVIEW Data Acquisition VIs*, in the *LabVIEW Function and VI Reference Manual*, or to the LabVIEW *Online Reference*, available by selecting **Help»Online Reference**.

# 6

# One-Stop Single-Point Acquisition

This chapter shows you how to acquire one data point from a single channel and then one data point from each of several channels using LabVIEW.

## Single-Channel, Single-Point Analog Input

A single-channel, single-point analog input is an immediate, nonbuffered operation. In other words, the software reads one value from an input channel and immediately returns the value to you. This operation does not require any buffering or timing. You should use single-channel, single-point analog input when you need one data point from one channel. An example of this would be if you periodically needed to monitor the fluid level in a tank. You can connect the transducer that produces a voltage representing the fluid level to a single channel on your DAQ device and initiate a single-channel, single-point acquisition whenever you want to know the fluid level.

For most basic operations, use the AI Sample Channel VI, located in the **Functions»DAQ»Analog Input** palette. The Easy Analog Input VI, AI Sample Channel, measures the signal attached to the channel you specify on your DAQ device and returns the scaled value. Figure 6-1 shows how to wire this VI.

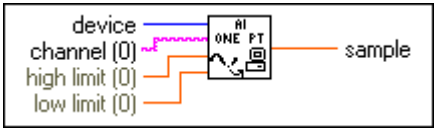



**Figure 6-1.** AI Sample Channel VI

**Note** If you set up your channel in the DAQ Channel Wizard, you do not need to enter the device or input limits. Instead, enter a channel name in the channel input, and the value returned is relative to the physical units you specified for that channel in the DAQ Channel Wizard. If you specify the input limits, they are treated as being relative to the physical units of the channel. LabVIEW ignores the device input when channel names are used. This principle applies throughout this manual.

Figure 6-2 shows how you program the Acquire 1 Point from 1 Channel VI, located in `labview\examples\daq\anlogin\anlogin.llb`, using the AI Sample Channel VI to acquire data.

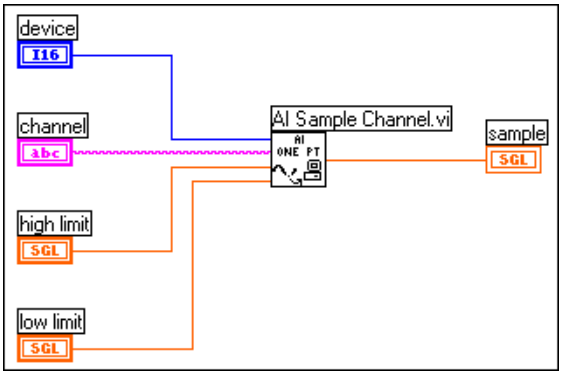



**Figure 6-2.** Acquiring Data Using the Acquire 1 Point from 1 Channel VI

The Acquire 1 Point from 1 Channel VI initiates an A/D conversion on the DAQ device and returns the scaled value as an output. The **high limit** is the highest expected level of the signals you want to measure. The **low limit** is the lowest expected level of the signals you want to measure. If you want to acquire multiple points from a single channel, see Chapter 7, *Buffering Your Way through Waveform Acquisition*.

Single-channel acquisition makes acquiring one channel very basic, but what if you need to take more than one channel sample? For example, you might need to monitor the temperature of the fluid as well as the fluid level of the tank. In this case, two transducers must be monitored. You can monitor both transducers using a multiple-channel, single-point acquisition in LabVIEW.

# Multiple-Channel Single-Point Analog Input

With a multiple-channel, single-point read (or scan), LabVIEW returns the value on several channels at once. Use this type of operation when you have multiple transducers to monitor and you want to retrieve data from each transducer at the same time. Your DAQ device executes a scan across each of the specified channels and returns the values when finished. Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Function and VI Reference Manual*, for the number of channels your device can scan at one time. You also can refer to the LabVIEW *Online Reference*, available by selecting **Help»Online Reference**.

The Easy I/O VI, AI Sample Channels, acquires single values from multiple channels. The AI Sample Channels VI performs a single A/D conversion on the specified channels and returns the scaled values in a 1-dimensional (1D) array. The expected range for all the signals, specified by **high limit** and **low limit** inputs, applies to all the channels. Figure 6-3 shows how to acquire a signal from multiple channels with this VI.

**Note** Remember to use commas to delimit individual channels in the channel string. Use a colon to indicate an inclusive list of channels.

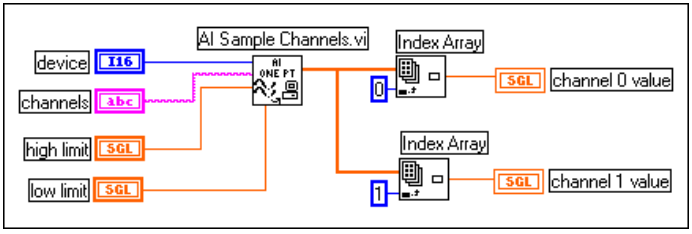



**Figure 6-3.** Acquiring a Voltage from Multiple Channels with the AI Sample Channels VI

You can benefit from using the Easy Analog Input VIs because you only need one icon in your diagram to perform the task, there are only a few basic inputs to the VIs, and the VIs have built-in error checking. However, the lack of programming flexibility with these VIs can be a limitation. Because Easy VIs have only a few inputs, you cannot implement some of the more detailed features of DAQ devices, such as triggering or interval scanning. In addition, these VIs always reconfigure at start-up. When you need a high-speed or efficiently run program, these configurations can slow down processing time.

When you need speed and more efficiency, use the Intermediate VIs, which configure an acquisition only once and then continually acquire data without re-configuring. The Intermediate VIs also offer more error handling control, more hardware functionality, and efficiency in developing your application than the Easy VIs. You typically use the Intermediate VIs to perform buffered acquisitions. You can read more about buffered acquisitions in Chapter 7, *Buffering Your Way through Waveform Acquisition*. The Intermediate Analog Input VI, AI Single Scan VI, does multiple-channel, single-point acquisitions, as shown in Figure 6-4.

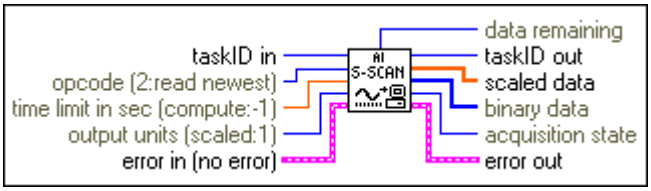



**Figure 6-4.** The AI Single Scan VI Help Diagram

The AI Single Scan VI returns one scan of data. You can also use this VI to read only one point if you specify one channel. Use this VI only in conjunction with the AI Config VI.

Figure 6-5 shows a simplified block diagram for non-buffered applications. LabVIEW calls the AI Config VI, which configures the channels, selects the **input limits** (the **high limit** and **low limit** inputs in the Easy VIs), and generates a **taskID**. The program passes the **taskID** and the error cluster to the AI Single Scan VI, which returns the data in an array (one point for each channel specified).

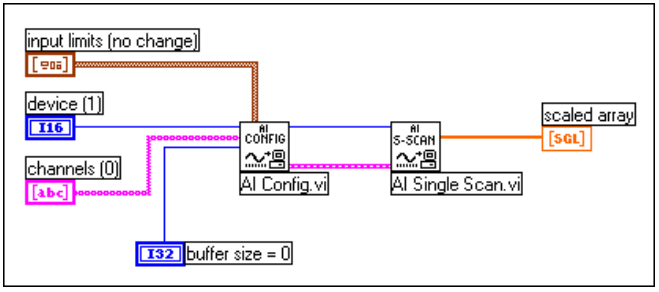



**Figure 6-5.** Using the Intermediate VIs for a Basic Non-Buffered Application

Figure 6-6 shows how you can program the AI Config and AI Single Scan VIs to perform a series of single scans by using software timing (a While Loop) and processing each scan. This example shows the Cont Acquire&Chart (immediate) VI, which you can find in `labview\examples\daq\anlogin\anlogin.llb`.

The advantage to using the intermediate-level VIs is that you do not have to configure the channels every time you want to acquire data as you do when using the Easy VIs. To call the AI Config VI only once, put it outside of the While Loop in your program. The AI Config VI configures channels, selects a high/low limit, and generates a **taskID**. Then, the AI Config VI passes the **taskID** and error cluster into the While Loop, where LabVIEW calls the AI Single Scan VI to retrieve a scan. The program then passes the returned data to the My Single-Scan Processing VI. With this VI, you can program whatever processing needs your application calls for, such as looking for a limit to be exceeded. The VI then passes the data through the build array function to a waveform chart for display on the front panel. The Wait Until Next ms Multiple (metronome) function controls the loop timing. You enter a scan rate, the application converts the value into milliseconds and passes the converted value to the Wait Until Next ms Multiple function. The loop then executes at the rate of scanning. The loop ends when you press the stop button or an when error occurs. Once the loop finishes, the Simple Error Handler VI displays any errors that occurred.

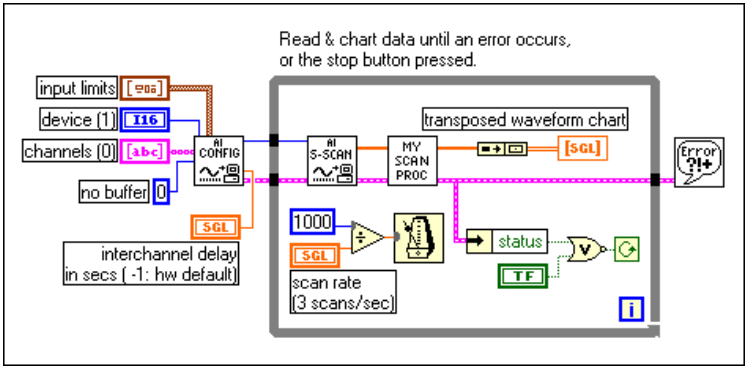



**Figure 6-6.** The Cont Acq&Chart (Immediate) VI Block Diagram

The previous examples use software-timed acquisition. With this type of acquisition, the CPU system clock controls the rate at which you acquire data. Your system clock can be interrupted by user interaction, so if you do not need a precise acquisition rate, use software-timed analog input.

# Using Analog Input/Output Control Loops

When you want to output analog data after receiving some analog input data, use analog input/output (I/O) control loops. With control loops, this process is repeated over and over again.

The single-point analog input and output VIs support several analog I/O control loops at once because you can acquire analog inputs from several different channels in one scan, and write all the analog output values with one update. You perform a single analog input call, process the analog output values for each channel and then perform a single analog output call to update all the output channels.

The following sections describe the two different types of analog I/O control loop techniques: *software-timed* and *hardware-timed analog I/O*.

## Using Software-Timed Analog I/O Control Loops

With software-timed analog control loops the analog acquisition rate and subsequent control loop rate are controlled by a software timer such as the Wait Until Next ms Multiple timer. The acquisition is performed during each loop iteration when the AI Single Scan VI is called and the control loop is executed once for each time interval. Your loop timing can be interrupted by any user interaction, which means your acquisition rate is not as consistent as that which can be achieved through hardware-timed control loops. Generally, if you do not need a precise acquisition rate for your control loop, software timing is appropriate.

Besides user interaction, a large number or large-sized front panel indicators, like charts and graphs, affect control loop rates. Refreshing the monitor screen interrupts the system clock, which controls loop rates. Therefore, you should keep the number of charts and graphs to a minimum when you are using software-timed control loops.

An example of software-timed control loops is the Analog IO Control Loop (immed) VI located in `labview\examples\daq\anlog_io\anlog_io.llb`.

The following diagram shows how to perform software-timed analog I/O using the AI Read One Scan and AO Write One Update VIs.

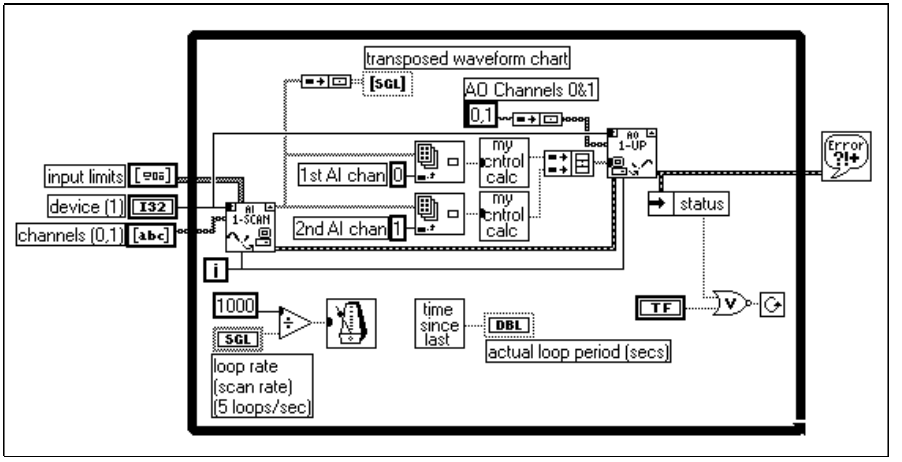



**Figure 6-7.** Software-Timed Analog I/O

The AI Read One Scan VI configures your DAQ device to acquire data from analog input channels 0 and 1. Once your program acquires a data point from channels 0 and 1, it performs calculations on the data and outputs the results through analog output channels 0 and 1. Because the iteration count is connected to the AI Read One Scan and AO Write One Update VIs, the application configures the DAQ device for analog input and output only on the first iteration of the loop. The loop rate as well as the acquisition rate is specified by **loop rate**. The reason why the **actual loop period** is important is because user interaction affects the loop and acquisition rate. For example, pressing the mouse button interrupts the system clock, which controls the loop rate. If your analog acquisition rate for control loops does not need to be consistent, then use software-timed control loops.

For more control examples, refer to the VIs located in `examples\daq\solution\control.llb`.

## Using Hardware-Timed Analog I/O Control Loops

For more precise timing of your control loops, and more precise analog input scan rate, use hardware-timed control loops.

An example of hardware-timed, non-buffered control loops is the Analog IO Control Loop (hw timed) VI located in `labview\examples\daq\anlog_io\anlog_io.llb`.

With hardware-timed control loops, your acquisition is not interrupted by user interaction. Hardware-timed analog input automatically places the data in your DAQ device FIFO buffer at an interval determined by the analog input scan rate. You can synchronize your control loop diagram to this precise analog input scan rate by repeatedly calling the AI Single Scan VI to read the oldest data in the FIFO buffer.

The AI Single Scan VI returns as soon as the next scan has been acquired by the DAQ device. If more than one scan is stored in the DAQ device FIFO buffer when the AI Single Scan VI is called, then the LabVIEW diagram was not able to keep up with the acquisition rate. You can detect this by monitoring the data remaining output of the AI Single Scan VI. In other words, you have missed at least one control loop interval. This indicates that your software overhead is preventing you from keeping up with your hardware-timed loop rate. In Figure 6-8, the **loop too slow** Boolean indicator is set to TRUE whenever this occurs.

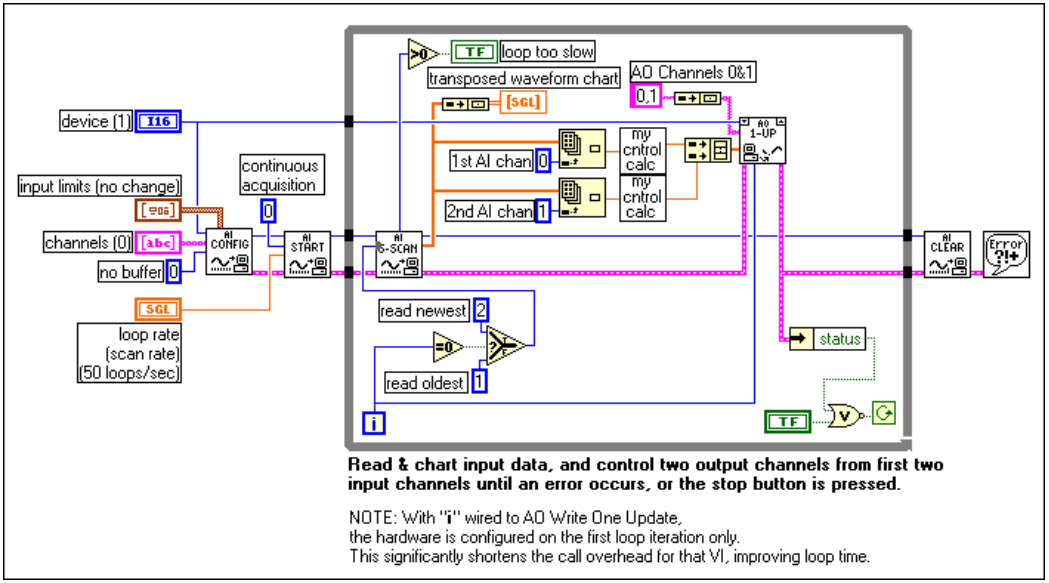



**Figure 6-8.** Analog IO Control Loop (HW-Timed) VI Block Diagram

In this diagram, the AI Config VI configures the device to acquire data on channels 0 and 1. The application does not use a buffer created in CPU memory, but instead uses the DAQ device FIFO buffer. **input limits** (also known as *limit settings*) affects the expected range of the input signals. For more information on input limits (limit settings), refer to Chapter 3, *Basic LabVIEW Data Acquisition Concepts*. The AI Start VI begins the analog

acquisition at the **loop rate** (scan rate) parameter. On the first iteration of the loop, the AI Single Scan VI reads the newest data in the FIFO buffer. Some data may have been acquired between the execution of the AI Start and the AI Single Scan VIs. On the first iteration of the loop, the application reads the latest data acquired between the AI Start and the AI Single Scan VIs. On every subsequent iteration of the loop, the application reads the oldest data in the FIFO buffer, which is the next acquired point in the FIFO buffer.

If more than one value was stored in the DAQ device FIFO buffer when you read it, your application was not able to keep up with the control loop acquisition and you have not responded with one control loop interval. This eventually leads to an error condition, which makes the loops complete. After the application completes analog acquisition and generation, then the AI Clear VI clears the analog input task.

Figure 6-8 also includes a waveform chart in the control loop. This reduces your maximum loop rate. You can speed up the maximum rate of the control loop by removing this graph indicator.

You easily can add other processing to your analog I/O control loop by putting the analog input, control loop calculations and analog output in the first frame of a sequence inside the loop, and additional processing in subsequent frames of the sequence. Keep in mind that this additional processing must be less than your control loop interval. Otherwise, you will not be able to keep up with your control loop rate.

## Improving Control Loop Performance

There are some performance issues you should take into account if you plan to have other VIs or loops execute in parallel with your hardware-timed control loop. When you call the AI Single Scan VI in a hardware-timed control loop, the VI waits until the next scan is acquired before returning, which means that the CPU is waiting inside the NI-DAQ driver until the scan is acquired. Consequently, if you try to run other LabVIEW VIs or while loops in the same diagram in parallel with your hardware-timed control loop, they may run more slowly or intermittently. You can reduce this problem by putting a software delay, with the Wait (ms) VI, at the end of your loop after you write your analog output values. Your other LabVIEW VIs and loops can then execute during this time.

Another good technique is to poll for your analog input without waiting in the driver. You can set the AI Single Scan VI **time limit in sec** to `0`. Then, the VI reads the DAQ Device FIFO buffer and returns immediately, regardless of whether the next scan was acquired. The AI Single Scan VI

**scaled data** output array is empty if the scan was not yet acquired. Poll for your analog input by using a Wait (ms) or Wait Until Next ms Multiple function together with the AI Single Scan VI in a While Loop within your control loop diagram. Set the wait time smaller than your control loop interval (at least half as small). If the **scaled data** output array is not empty, exit the polling loop passing out the **scaled data** array and execute the rest of your control loop diagram. This method does not return data as soon as the scan has been acquired, as in the example described previously, but provides ample time for other VIs and loops to execute. This method is a good technique for balancing the CPU load between several loops and VIs running in parallel.

For more control examples, refer to the VIs located in `examples\daq\solution\control.llb`.

# 7

# Buffering Your Way through Waveform Acquisition

If you want to take more than one reading on one or more channels, there are two techniques you can use depending on what you want to do with the data after you acquire it. This chapter reviews these different methods and explains how LabVIEW stores the acquired data with each method. You will discover which method you should use by answering the following questions:

- Do you want to analyze your data as it is being acquired or after it has been acquired?
- Do you want to acquire a predetermined or indefinite number of data points?

If you want to analyze your data as it is being measured and the number of data points does not matter, read the *Do You Need to Access Your Data during Acquisition?* section in this chapter. If you acquire a predetermined number of data points and you want to analyze the data after it has been acquired, refer to the *Can You Wait for Your Data?* section in this chapter. Also, throughout the chapter there are some basic examples of some common DAQ applications that use these two methods.

## Can You Wait for Your Data?

One way to acquire multiple data points for one or more channels is to use the non-buffered methods described in the previous chapter in a repetitive manner. For example, you could compare this method to a trip to the grocery store. You need to get 20 items from the store, but because you can't carry all 20 items at once, you decide you must make 20 separate trips to the store. Grocery shopping in this manner would be very inefficient and time consuming. The same applies for when you are acquiring a single data point from one or more channels over and over. Also, with this method of acquisition, you do not have accurate control over the time between each sample or channel. Going back to the example of grocery shopping, it would be much more efficient to use a shopping bag to hold all 20 food items at once, so that you only have to make one trip. In the same sense,

you can use a data buffer in computer memory as your shopping bag with which you acquire data.

With buffered I/O, LabVIEW transfers data taken at timed intervals from a DAQ device to a data buffer in memory. Figure 7-1 illustrates how the data fills up the buffer only once, however the overall size of the buffer is specified in your VI. In this illustration, think of **N** as the number of scans or updates the buffer can hold, and **T** as the trigger occurrence whether the trigger is because of an external signal or the start of the execution of your VI. Refer to Chapter 8, *Controlling Your Acquisition with Triggers*, for more descriptions on triggering your acquisition from another signal.

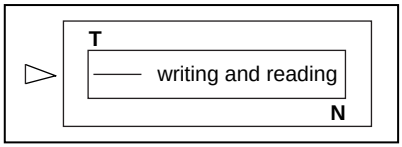



**Figure 7-1.** How Buffers Work

In your VI, you must specify the number of samples to be taken and the number of channels from which LabVIEW will take the samples. From this information, LabVIEW allocates a buffer in memory to hold a number of data points equal to the number of samples per channel multiplied by the number of channels. As the data acquisition continues, the buffer fills with the data. However, the data may not actually be accessible until LabVIEW acquires all the samples (N). Once the data acquisition is complete, the data in the buffer can be analyzed, stored to disk, or displayed to the screen by your VI.

## Acquiring a Single Waveform

You can acquire a waveform from a single channel by using the AI Acquire Waveform VI, shown in Figure 7-2. You can find this VI in **Functions»DAQ»Analog Input**. Because AI Acquire Waveform is an Easy Analog Input VI, it has the minimal number of inputs needed to acquire a waveform from a single channel. These minimal inputs are the **device**, **channel string**, **number of samples** from the channel, and the **sample rate**. You can programmatically set the gain by setting the **high limit** and the **low limit**. Using only the minimal set of inputs makes programming the VI easier, but the VI lacks more advanced capabilities, such as triggering. Built-in error handling is another useful feature of the Easy VIs. If an error occurs, the program stops running and notifies you with a dialog box explaining the error.

**Note** If you set up your channel in the DAQ Channel Wizard, you do not need to enter the device or input limits. Instead, enter a channel name in the channel input, and the value returned is relative to the physical units you specify for that channel in the DAQ Channel Wizard. If input limits are specified, they also are treated as relative to the physical units of the channel. LabVIEW ignores the device input when channel names are used. This principal applies throughout this manual.

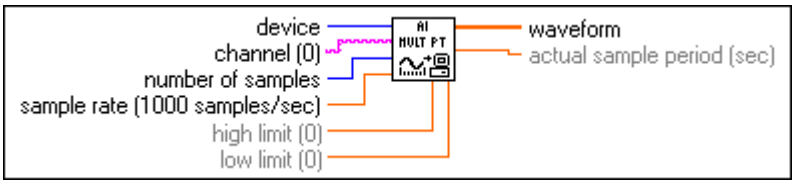



**Figure 7-2.** The AI Acquire Waveform VI

## Acquiring Multiple Waveforms

You can acquire more than one waveform at a time with another of the Easy Analog Input VIs, AI Acquire Waveforms, shown in Figure 7-3. This VI also has a minimal set of inputs, but it allows inputs of more than one channel to read and returns data from all channels read.

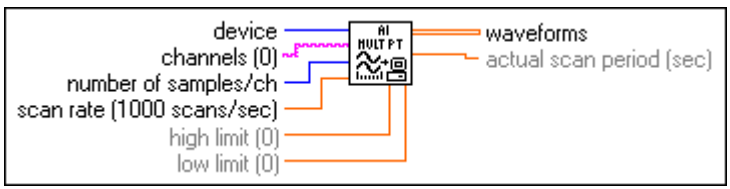



**Figure 7-3.** The AI Acquire Waveforms VI

The **channel** input for this VI is a string where you can enter a list of channels. Refer to Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, for more information on channel specification in LabVIEW. LabVIEW outputs a two-dimensional (2D) array in the **waveforms** output for this VI, where each channel has a different column and the samples are in each row. See Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, for more information on how data is organized for analog applications. You can set the **high limit** and **low limit** inputs for all the channels to the same value. For more information on gain specifications, refer to Chapter 3, *Basic LabVIEW Data Acquisition Concepts*. Like the other Easy VIs, you cannot use any advanced programming features with the AI Acquire Waveforms VI. The built-in error checking of this VI alerts you to any errors that occur in the program.

You also can acquire multiple waveforms using the Intermediate VIs. The Intermediate VIs provide more control over your data acquisition processes, like being able to read any part of the buffer. An example similar to Figure 7-4 is the Acquire N Scans VI, located in `labview\examples\daq\anlogin\anlogin.llb`. With these Intermediate Analog Input VIs, you must wire a **taskID** to identify the DAQ operation and the set of channels used in the acquisition and to make sure the VIs execute in the correct order.

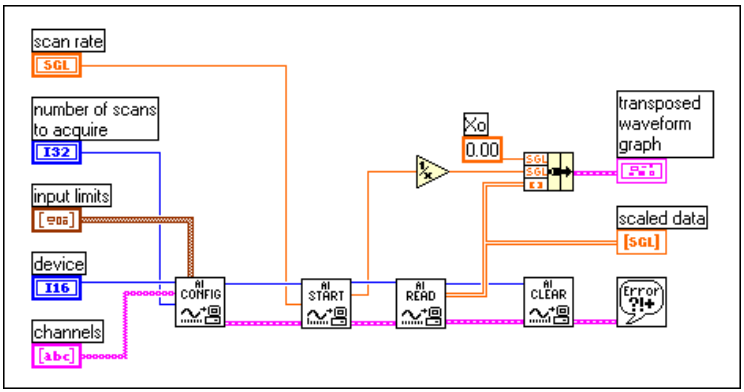



**Figure 7-4.** Using the Intermediate VIs to Acquire Multiple Waveforms

With the Intermediate VIs, can you configure triggering, coupling, acquisition timing, retrieval, and additional hardware, and control when each step of the data acquisition process occurs. With the AI Config VI, you can configure the different parameters of the acquisition, such as the channels to be read and the size of the buffer to use. In the AI Start VI, you specify parameters used in your program to start the acquisition, such as number of scans to acquire, the rate at which your VI takes the data, and the trigger settings. In the AI Read VI, you specify parameters to retrieve the data from the data acquisition buffer. Then, your application calls the AI Clear VI to deallocate all buffers and other resources used for the acquisition by invalidating the **taskID**. If an error occurs in any of these VIs, your program passes the error through the remaining VIs to the Simple Error Handler VI, which notifies you of the error.

For many DAQ devices, the same ADC samples many channels instead of only one. The maximum sampling rate per channel is

$$\frac{\textit{maximum sampling rate}}{\textit{number of channels}}$$

The scan rate input in all the VIs described above is the same as the sampling rate per channel. To figure out your maximum scan rate, you must divide the maximum sampling rate by the number of channels. In Appendix B, *Hardware Capabilities*, in the *LabVIEW Function and VI Reference Manual*, maximum sampling rates are listed for each DAQ device. You also can refer to the LabVIEW *Online Reference*, available by selecting **Help»Online Reference**.

**Note** When using the NB-A2100 or the NB-A2150 boards, specifying an odd buffer size or an odd number of samples when acquiring data with one channel results in **`-10089 badTotalCountError`**. To avoid this error, specify an even number of samples and throw away the extra sample.

# Simple-Buffered Analog Input Examples

Following are several different examples of simple-buffered analog input.

## Simple-Buffered Analog Input with Graphing

Figure 7-5 show how you can use the AI Acquire Waveforms VI to acquire two waveforms from channels 0 and 1 and then display the waveforms on separate graphs. This type of VI is useful in comparing two or more waveforms, or in analyzing how a signal looks before and after going through a system. In this illustration, 1,000 scans of channels 0 and 1 are taken at the rate of 5,000 scans per second. The **actual scan period** output displays in the actual timebase on the X-Axis of the graphs. Remember that each column of the 2D array contains the information for each channel.

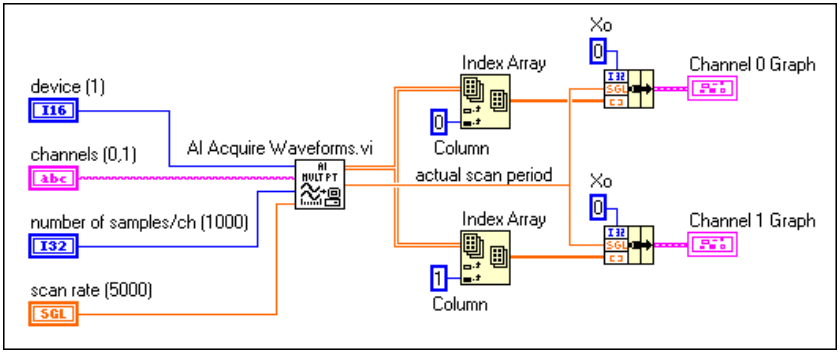



**Figure 7-5.** Simple Buffered Analog Input Example

If you want to display the data on the same graph, look again at the Acquire N Scans example VI, found in `labview\examples\daq\anlogin\anlogin.llb`. Figure 7-6 shows a simple buffered input application that uses graphing.

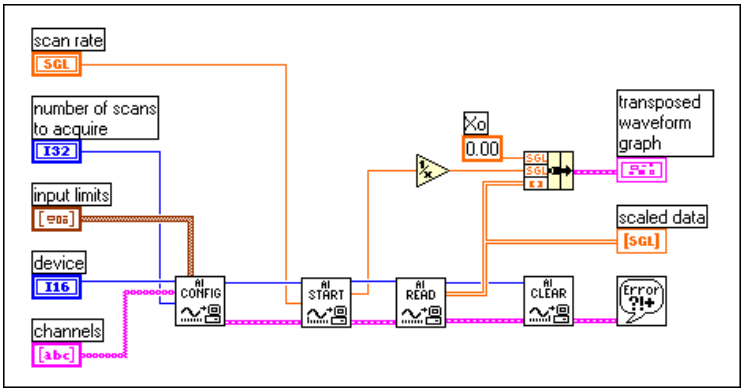



**Figure 7-6.** Simple Buffered Analog Input with Graphing

For a 2D array to be displayed on a waveform graph, each row of data must represent a single plot. This is because waveform graphs are in row-major order. Because the channel data is in each column, you must transpose the 2D array. Transposing the array can be done easily by popping up on the front panel of the graph and choosing **Transpose Array**.

## Simple-Buffered Analog Input with Multiple Starts

In some cases, you might not want to acquire contiguous data, like in an oscilloscope application. In this case, you would only want to take a specified number of samples as a snapshot of what the input looks like periodically. For an example using the Intermediate VIs, open the Acquire N-Multi-Start VI found in `labview\examples\daq\anlogin\anlogin.llb`. The Acquire N-Multi-Start VI, shown in Figure 7-7, is similar to the Acquire N Scans example, except the acquisition only occurs each time the start button on the front panel is pressed.

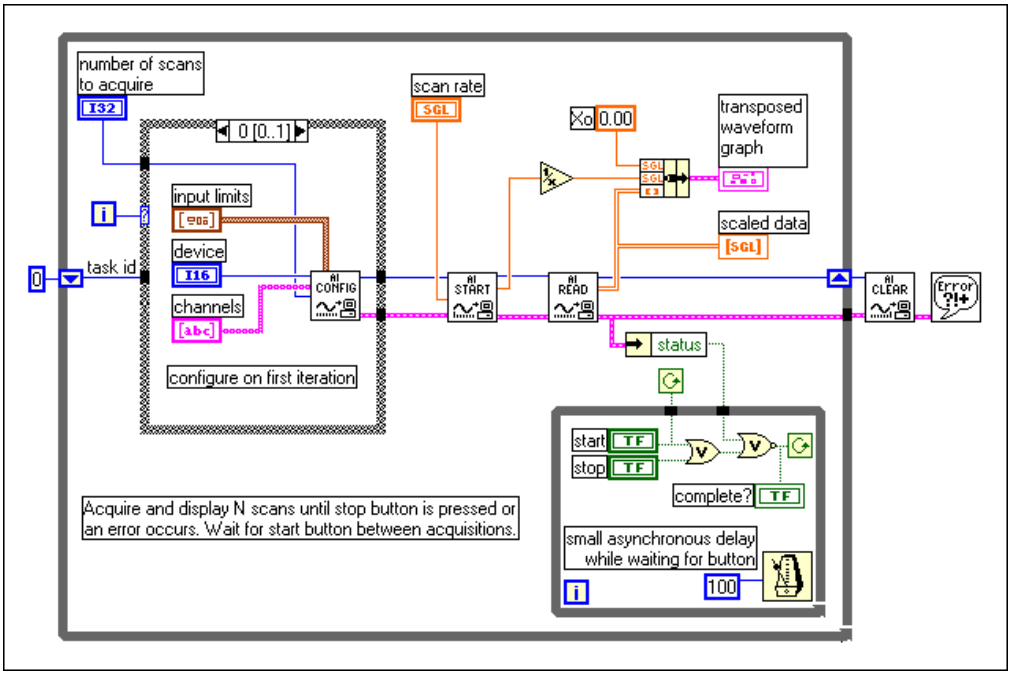



**Figure 7-7.** Taking a Specified Number of Samples with the Intermediate VIs

This example is similar to the standard simple buffered analog input VI, but now both the AI Start and AI Read VIs are in a While Loop, which means the program takes a number of samples every time the While Loop iterates.

**Note** The AI Read VI returns 1,000 samples, taken at 5,000 samples per second, every time the While Loop iterates. However, the duration of the iterations of the While Loop can vary greatly. This means that, with this VI, you can control the rate at which samples are taken, but you may not be able to designate exactly when your application starts acquiring each set of data. If this start-up timing is important to your program, read the *Do You Need to Access Your Data during Acquisition?* section in this chapter to see how to control acquisition start-up times.

## Simple-Buffered Analog Input with a Write to Spreadsheet File

If you want to write the acquired data to a file, there are many file formats in which you can store the data. The spreadsheet file format is used most often because you can read it using most spreadsheet applications for later data graphing and analysis. In LabVIEW, you can use VIs to send data to a file in spreadsheet format or read back data from such a file. You can locate these VIs in **Functions»File I/O**. The VI used in this example is the Write to Spreadsheet File VI, shown in Figure 7-8. In this exercise, the Intermediate analog input VIs acquire an array of data, graph the data using the actual sample period for the x-axis timebase, and create a spreadsheet file containing the data.

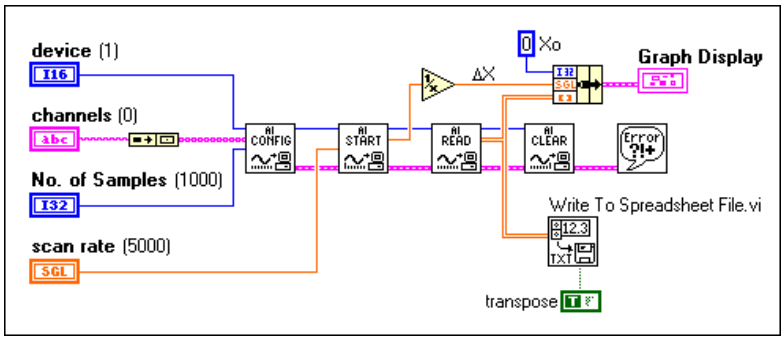



**Figure 7-8.** Writing to a Spreadsheet File after Acquisition

## Triggered Analog Input

For information on starting your acquisition with triggers, refer to Chapter 8, *Controlling Your Acquisition with Triggers*.

# Do You Need to Access Your Data during Acquisition?

You can apply the simple buffering techniques in many DAQ applications, but there are some applications where these techniques are not appropriate. If you need to acquire more data than your computer's memory can hold, or if you want to acquire data over long periods of time, you should not use these simple-buffered techniques. For these types of applications, you should set up a circular buffer to store acquired data in memory. In the previous section, buffered input was compared to shopping for groceries. You typically use a cart or bag (your buffer) to hold as many groceries (your acquired data) as possible, so that you only have to make one trip to the store. In this case, imagine that you must prepare a meal and you are unable to go shopping—yet periodically you need things from the store for

your recipe. If you send someone else to the store for you, you can continue to prepare dinner while someone else retrieves the other items you need. You can compare this scenario to circular-buffered data acquisition, shown in Figure 7-9. Using a circular buffer, you can set up your device to continuously acquire data in the background while LabVIEW retrieves the acquired data.

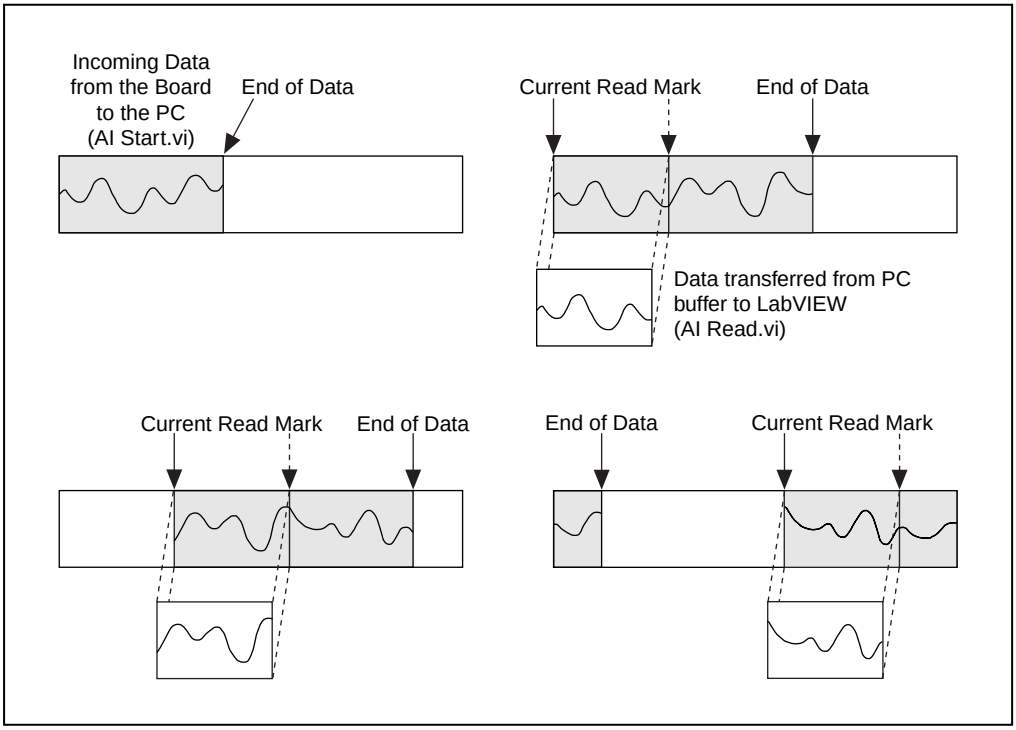



**Figure 7-9.** How a Circular Buffer Works

A circular buffer differs from a simple buffer only in how LabVIEW places the data into it, and retrieves data from it. A circular buffer is filled with data, just as a simple buffer. However, when it gets to the end of the buffer, it returns to the beginning and fills up the same buffer again. This means data can read continuously into computer memory, but only a defined amount of memory can be used. Your VI must retrieve data in blocks, from one location in the buffer, while the data enters the circular buffer at a different location, so that unread data is not overwritten by newer data. Because of the buffer maintenance, you can only use the Intermediate or Advanced VIs with this type of data acquisition.

While a circular buffer works well in many applications, there are two possible problems that can occur with this type of acquisition: Your VI could try to retrieve data from the buffer faster than data is placed into it, or your VI might not retrieve data from the buffer fast enough before LabVIEW overwrites the data into the buffer. When your VI tries to read data from the buffer that has not yet been collected, LabVIEW waits for the data your VI requested to be acquired and then returns the data. If your VI does not read the data from the circular buffer fast enough, the VI sends back an error, advising you that the data that you retrieved from the buffer is overwritten data.

## Continuously Acquiring Data from Multiple Channels

You can acquire time-sampled data continuously from one or more channels with the Intermediate VIs. An example using these VIs is the Acquire & Process N Scans VI, found in `labview\examples\daq\anlogin\anlogin.llb`. This example is shown in Figure 7-10. There are inputs for setting the channels, size of the circular buffer, scan rate, and the number of samples to retrieve from the circular buffer each time. This VI defaults to a **input buffer size** of 2,000 samples and 1,000 **number of scans to read at a time**, which means the VI reads in half of the buffer's data while the VI fills the second half of the buffer with new data.

**Note** The **number of scans to read** can be any number less than the input buffer size.

If you do not retrieve data from the circular buffer fast enough, your unread data will be overwritten by newer data. You can resolve this problem in one of three ways: by adjusting the **input buffer size**, **scan rate**, or the **number of scans to read at a time** parameters. If your program overwrites data in the buffer, then data is coming into the buffer faster than your VI can read all of the previous buffer data, and LabVIEW returns an error code **`-10846 overWriteError`**. You can increase the size of the buffer so that it takes longer to fill up, which leaves your VI with more time to read data from it. If you slow down the **scan rate**, you reduce the speed at which the buffer fills up, which also leaves more time for your program to retrieve data. You can also increase the **number of scans to read at a time**, which will retrieve more data out of the buffer each time and effectively reduce the number of times to access the buffer before it becomes full. Check the output **scan backlog** to see how many data values remain in the circular buffer after the read.

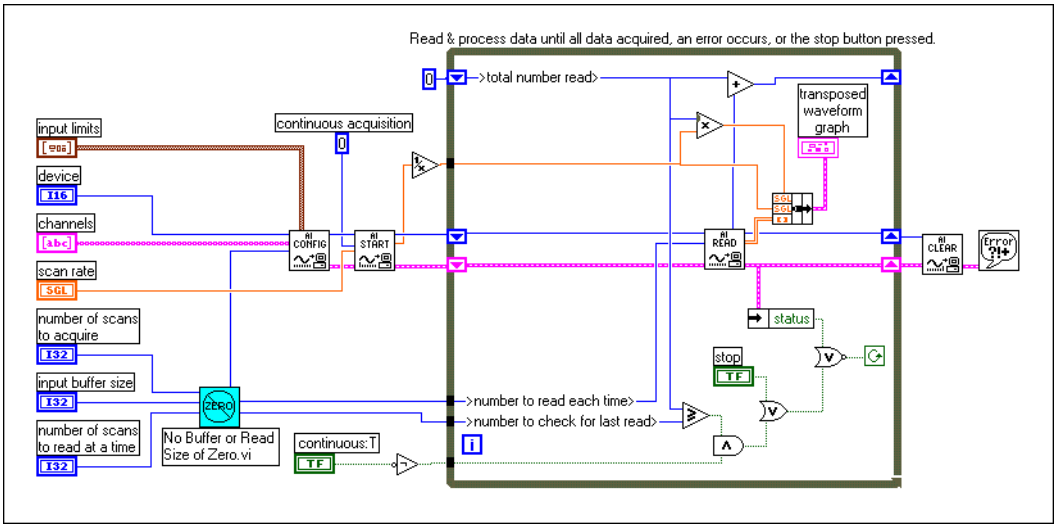



**Figure 7-10.** Continuously Acquiring Data with the Intermediate VIs

Because this uses Intermediate VIs, you also can control parameters such as triggering, coupling, and additional hardware.

## Asynchronous Continuous Acquisition Using DAQ Occurrences

The main advantage of the last section is that you are free to manipulate your data between calls to the AI Read VI. One limitation, however, is that the acquisition is synchronous. This means that once you call the AI Read VI, you cannot perform any other tasks until the AI Read VI returns your acquired data. If your DAQ device is still busy collecting data, you will have to sit idle until it finishes.

If you need the efficiency of not having to wait for the AI Read VI, then asynchronous acquisition is for you. You can acquire asynchronous continuous data from multiple channels using the same intermediate DAQ VIs by adding DAQ Occurrences. Figure 7-11 shows an example of how to do this. This is the diagram of the Cont Acq&Chart (Async Occurrence) VI, located in `labview\examples\daq\anlogin\anlogin.llb`. Notice that it is very similar to Figure 7-10.

The difference is that here you will use the DAQ Occurrence Config VI and the Wait on Occurrence function to control the reads. The first DAQ Occurrence Config VI sets the DAQ Event. In this example the **DAQ Event** is to set the occurrence every time a number of scans is acquired equal to the value of general value *A*, where general value *A* is the

**number of scans to read at a time**. Inside the While Loop, the Wait on Occurrence function sleeps in the background until the chosen **DAQ Event** takes place. Notice that the timed out output from the Wait on Occurrence function is wired to the selection terminal of the Case structure that encloses the AI Read VI. This means that AI Read will not be called until the **number of scans to read at a time** have been acquired. The result is that the While Loop is effectively put to sleep, because you do not try to read the data until you know it has been acquired. This frees up processor time to do other tasks while you are waiting for the DAQ Event. If the DAQ Occurrence times out, the timed out output value would be TRUE, and AI Read would never be called. When your acquisition is complete, DAQ Occurrence is called again to clear all occurrences.

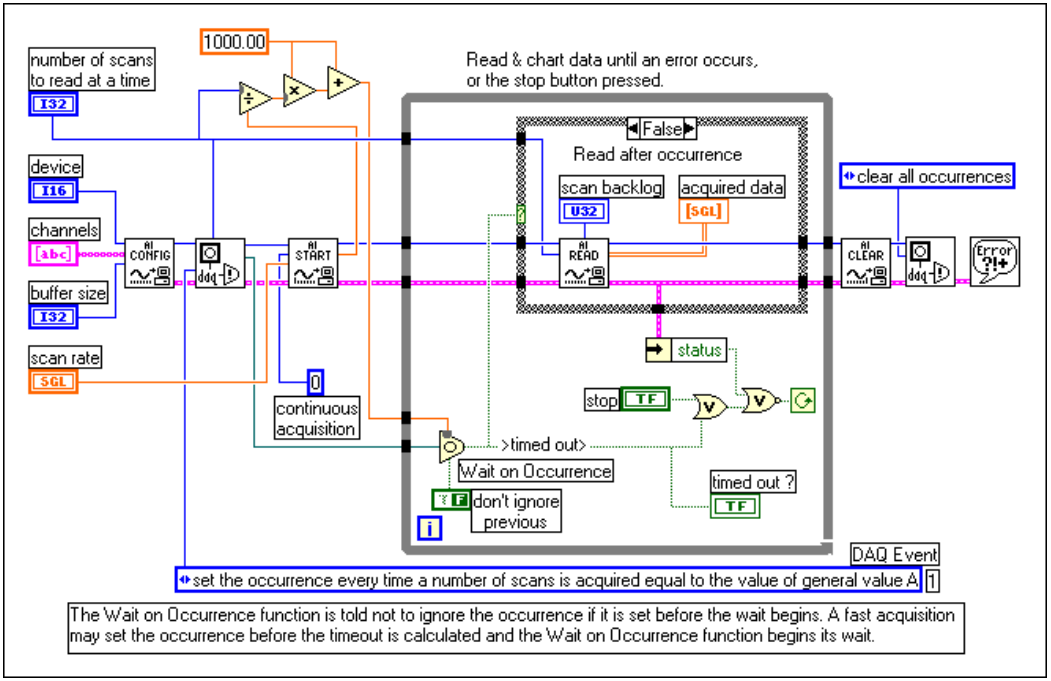



**Figure 7-11.** Continuous Acq&Chart (Async Occurrence) VI

# Circular-Buffered Analog Input Examples

The only differences between the simple-buffered applications and circular-buffered applications in the block diagram is the **number of scans to acquire** input of the AI Start VI is set to `0`, and you must call the AI Read VI repeatedly to retrieve your data. These changes can be applied to many of the examples in the previous section on simple buffered analog input, however this section reviews the basic circular-buffered analog input VI here and describes some other example VIs that are included with LabVIEW.

## Basic Circular-Buffered Analog Input

Figure 7-12 shows an example VI that brings data from channel 0 at a rate of 1,000 samples/s into a buffer that can hold 4,000 samples. This type of example might be handy if you wanted to watch the data from a channel over a long period of time, but you could not store all the data in memory at once. The AI Config VI sets up the channel specification and buffer size, then the AI Start VI initiates the background data acquisition and specifies the rate. Inside the While Loop, the AI Read VI repeatedly reads blocks of data from the buffer of a size equal to either 1,000 scans or the size of the **scan backlog**—whichever one is larger. The VI does this by using the Max & Min function to determine the larger of the two values. You do not have to use the Max & Min function in this way for the application to work, but the function helps control the size of the **scan backlog**, which is how many samples that are left over in the buffer. This VI continuously reads and displays the data from channel `0` until an error occurs or until you click the **Stop** button.

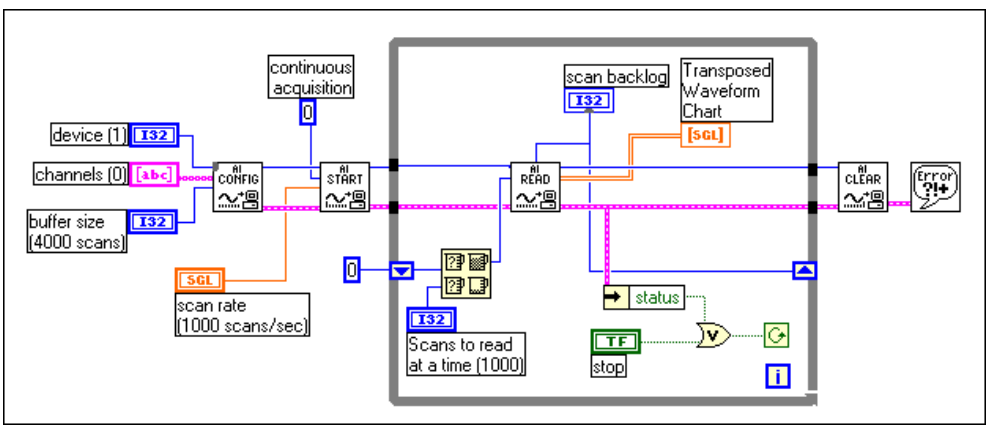



**Figure 7-12.** Basic Circular-Buffered Analog Input Using the Intermediate VIs

## Other Circular-Buffered Analog Input Examples

There are many other circular-buffered analog input VIs that are included with your LabVIEW application. The following list describes some of these VIs. You can find the first two VIs in `labview\examples\daq\anlogin\anlogin.llb` and the rest of the example VIs in `labview\examples\daq\anlogin\strmdsk.llb`. For information on how these examples work and how to modify them, open **Windows»Show VI Information** or open the Help window by choosing **Help»Show Help**.:

- **Cont Acq & Chart (buffered) VI**—Demonstrates circular-buffered analog input similarly to the previous example, but this VI includes other front panel inputs.
- **Cont Acq & Graph (buffered) VI**—Is similar to the Cont Acq & Chart (buffered) VI, except this VI displays data in a waveform graph.
- **Cont Acq to File (binary) VI**—Acquires data through circular-buffered analog input and stores it in a specified file as binary data. This process is more commonly called streaming to disk.
- **Cont Acq to File (scaled) VI**— Is similar to the previous binary VI, with the exception that this VI writes the acquired data to a file as scaled voltage readings rather than binary values.
- **Cont Acq to Spreadsheet File VI**—Continuously reads data that LabVIEW acquires in the circular buffer and stores this data to a specified file in spreadsheet format. You can view the data stored in a spreadsheet file by this VI in any spreadsheet application.

# Simultaneous Buffered Waveform Acquisition and Waveform Generation

You might discover that along with your analog input acquisition, you also would like to output analog data. If so, see Chapter 14, *Simultaneous Buffered Waveform Acquisition and Generation*.

# 8

# Controlling Your Acquisition with Triggers

The single-point and waveform acquisitions described in the previous sections start at random times relative to the data. But, there are times that you may need to be able to set your analog acquisition to start at a certain time. An example of this would be if you wanted to measure the temperature of an object after applying heat to it. An electrical thermometer sends a step voltage to your DAQ device after the heating completes. If you have no way to begin measuring data immediately after your device receives the step voltage, then you must acquire more points, some before the step voltage and some after it in order to capture the data you need. This solution is an inefficient use of computer memory and disk space, because you must allocate and use more than is necessary. Sometimes the data you need may be closer to the front of the buffer and other times it may be closer to the end of the buffer.

You can start an acquisition based on the condition or state of an analog or digital signal using a technique called *triggering*. Generally, a *trigger* is any event that starts data capture. There are two basic types of triggering—hardware and software triggering. In LabVIEW, you can use software triggering to start acquisitions or use it with an external device to perform hardware triggering.

## Hardware Triggering

Hardware triggering lets you set the start time of an acquisition and gather data at a known position in time relative to a trigger signal. External devices produce hardware trigger signals. In LabVIEW, you specify the triggering conditions that must be reached before acquisition begins. When the conditions are met, the acquisition begins. You can also analyze the data before the trigger.

There are two types of hardware triggers: digital and analog. In the following two sections, you will learn about the necessary conditions to start an acquisition with a digital or an analog signal.

## Digital Triggering

A *digital trigger* is usually a transistor-transistor logic (TTL) signal having two discrete levels—a high and a low level. When moving from high to low or low to high, a digital edge is created. There are two types of edges: rising and falling. You can set your analog acquisition to start as a result of the rising or falling edge of your digital trigger signal.

In Figure 8-1, the acquisition begins after the falling edge of the digital trigger signal. Usually digital trigger signals are connected to STARTTRIG*, EXTTRIG*, DTRIG, EXT TRIG IN, or PFI pins on your DAQ device. If you want to know which pin your device has, check your hardware manual, or refer to the AI Trigger Config VI description in Chapter 18, *Advanced Analog Input VIs*, of the *LabVIEW Function and VI Reference Manual*. You also can refer to the LabVIEW *Online Reference*, available by selecting **Help»Online Reference**. The STARTTRIG* and EXTTRIG* pins, which have and asterisk after their names, regard a falling edge signal as a trigger. Make sure you account for this when specifying your triggering conditions.

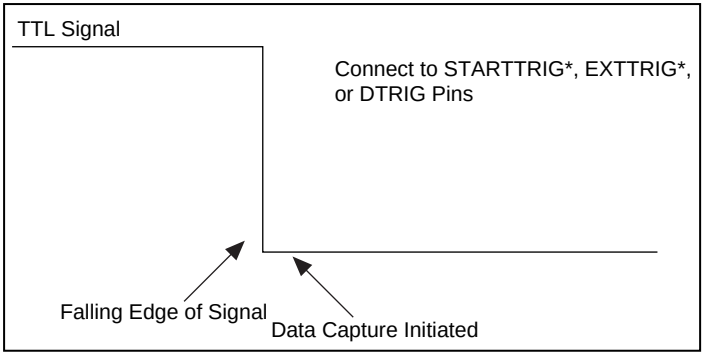



**Figure 8-1.** Diagram of a Digital Trigger

Figure 8-2 shows a timeline of how digital triggering works for post-triggered data acquisition. In this example, an external device sends a trigger, or TTL signal, to your DAQ device. As soon as your DAQ device receives the signal, and your trigger conditions are met, your device begins acquiring data.

With NI 406*x* hardware, start trigger pulses can be generated externally or internally. The following start trigger pulse sources apply:

- Software start trigger
- External trigger

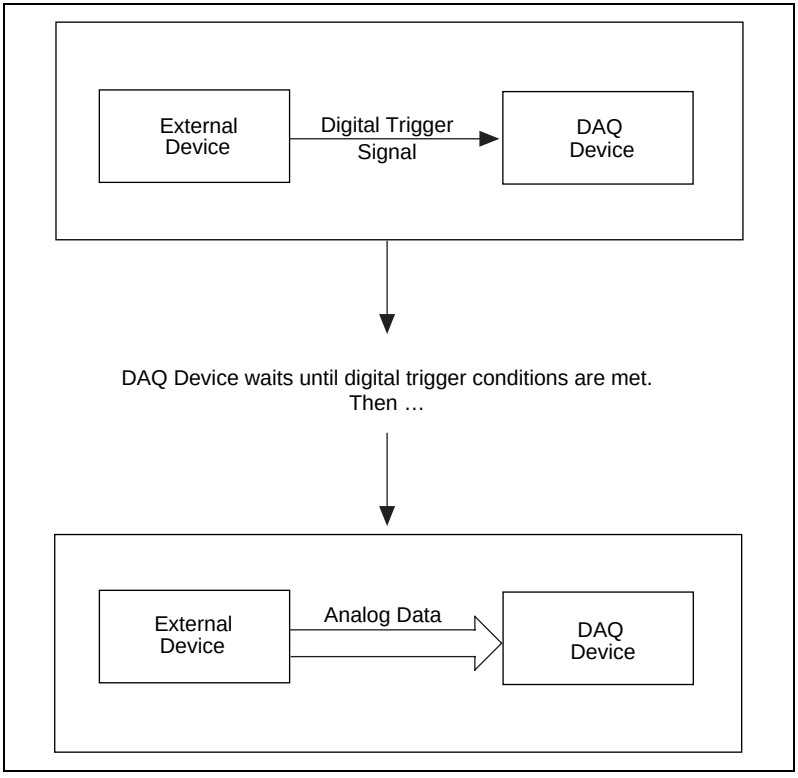



**Figure 8-2.** Digital Triggering with Your DAQ Device

## Digital Triggering Examples

An example of digital triggering is the Acquire N Scans Digital Trig VI, found in `labview\examples\daq\anlogin\anlogin.llb`. This VI, shown Figure 8-3, uses the Intermediate VIs to perform a buffered acquisition, where LabVIEW stores data in a memory buffer during acquisition. After the acquisition completes, the VI retrieves all the data from the memory buffer and displays it.

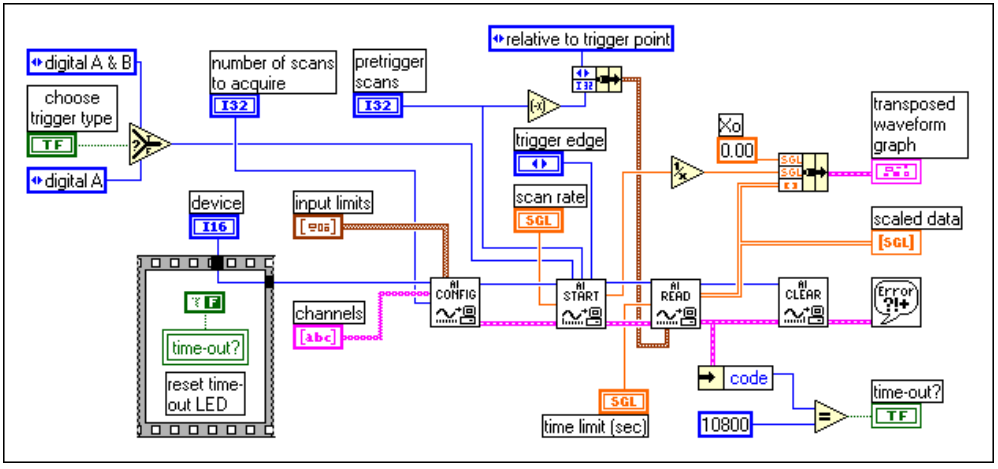



**Figure 8-3.** Block Diagram of the Acquire N Scans Digital Trig VI

For more information on buffered acquisitions, refer to Chapter 7, *Buffering Your Way through Waveform Acquisition*.

You must tell your device the conditions on which to start acquiring data.

For this example, the **choose trigger type** Boolean should be set to `START OR STOP TRIGGER`. You should only use the `START & STOP TRIGGER` when you have two triggers: start and stop. In addition, if you use a DAQ device with PFI lines (for example, E Series 5102 devices), you can specify the trigger signal condition in the **trigger channel** control in the **analog chan & level** cluster.

For more information on valid trigger channel names or two-trigger applications, refer to the AI Trigger Config VI description, in Chapter 18, *Advanced Analog Input VIs*, of the *LabVIEW Function and VI Reference Manual*, or to the LabVIEW *Online Reference*, available by selecting **Help»Online Reference**.

You can acquire data both before and after a digital trigger signal. If **pretrigger scans** is greater than 0, your device acquires data before the triggering conditions are met and subtracts the **pretrigger scans** value from the **number of scans to acquire** value to determine the number of scans to collect after the triggering conditions are met. If **pretrigger scans** is 0, you acquire the **number of scans to acquire** after the triggering conditions are met.

Before you start acquiring data, you must specify in the **trigger edge** input whether the acquisition is triggered on the rising or falling edge of the digital trigger signal. You also can specify a value for the **time limit**, the maximum amount of time the VI waits for the trigger and requested data.

## Digital Triggering Examples

The Acquire N Scans Digital Trig VI example holds the data in a memory buffer until your device completes the acquisition. The number of data points you need to acquire must be small enough to fit in memory. This VI only views and processes the information after the acquisition. To view and process information during the acquisition, use the Acquire & Proc N Scans-Trig VI, found in `labview\examples\daq\anlogin\anlogin.llb`. If you expect multiple digital trigger signals that start multiple acquisitions, use the VI, Acquire N-Multi-Digital Trig, located in `labview\examples\daq\anlogin\anlogin.llb`.

Figure 8-4 shows an example of using the digital trigger A to start a buffered acquisition.

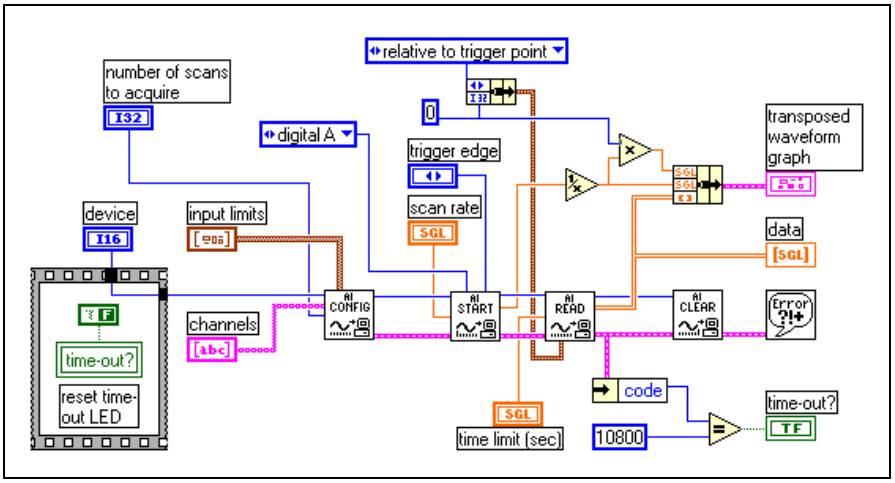



**Figure 8-4.** Block Diagram of a VI Acquiring Data On with Digital Trigger A

You connect analog trigger signals to the analog input channels—the same channels where you connect analog data. Your DAQ device monitors the analog trigger channel until trigger conditions are met. You configure the DAQ device to wait for a certain condition of the analog input signal, like the signal level or slope (either rising or falling). Once the device identifies the trigger conditions, it starts an acquisition.

**Note** If you are using channel names configured in the DAQ Channel Wizard, the signal level is treated as being relative to the physical units specified for the channel. For example, if you configure a channel called `temperature` to have a physical unit of `Deg. C`, the value you specify for the trigger signal level is relative to `Deg. C`. If you are not using channel names, the signal level is treated as volts.

In Figure 8-5, the analog trigger is set to start the data acquisition on the rising slope of the signal, when the signal reaches 3.2.

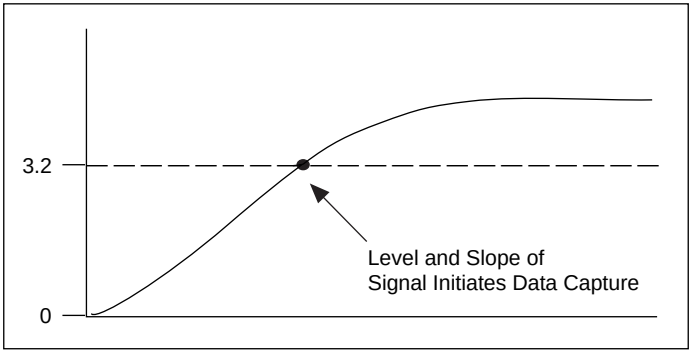



**Figure 8-5.** Diagram of an Analog Trigger

Figure 8-6 explains analog triggering for post-triggered data acquisition using a timeline. You configure your DAQ hardware in LabVIEW to begin taking data when the incoming signal is on the rising slope and when the amplitude reaches 3.2. Your DAQ device begins capturing data when the specified analog trigger conditions are met.

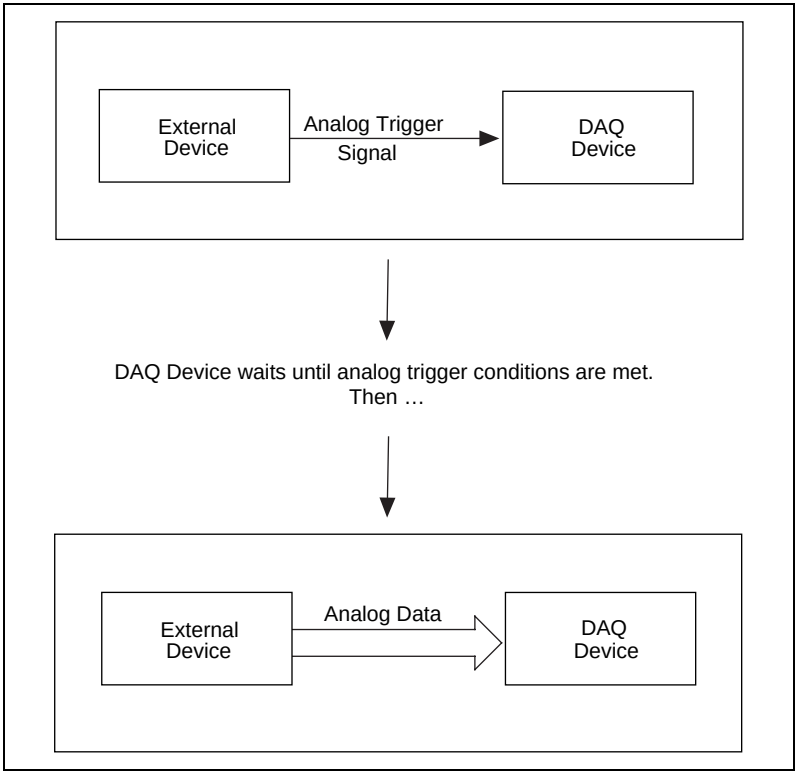



**Figure 8-6.** Analog Triggering with Your DAQ Device

## Analog Triggering Examples

A common example of analog triggering in LabVIEW is the Acquire N Scans Analog Hardware Trig VI, located in `labview\examples\daq\anlogin\anlogin.llb`. This VI, as shown in Figure 8-7, uses the Intermediate VIs to perform buffered acquisition, where data is stored in a memory buffer during acquisition. After the acquisition completes, the VI retrieves all the data from the memory buffer and displays it.

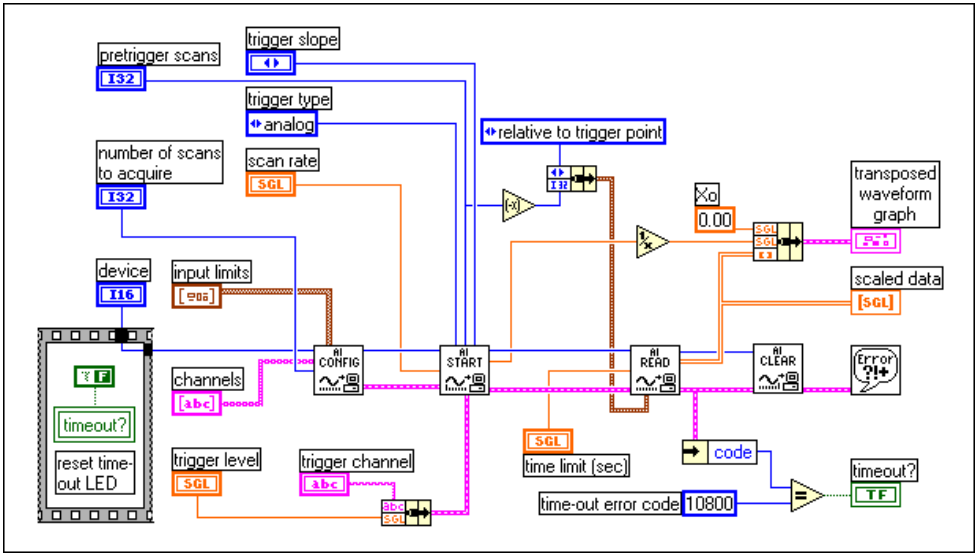



**Figure 8-7.** Block Diagram of the Acquire N Scans Analog Hardware Trig VI

For more information on buffered acquisition, read Chapter 7, *Buffering Your Way through Waveform Acquisition*.

You must tell your device the conditions on which to start acquiring data.

In LabVIEW, you can acquire data both before and after an analog trigger signal. If the **pretrigger scans** is greater than 0, your device acquires data before the triggering conditions and subtracts the **pretrigger scans** value from the **number of scans to acquire** value to determine the number of scans to collect after the triggering conditions are met. If **pretrigger scans** is 0, then the **number of scans to acquire** will be acquired after the triggering conditions are met.

Before you start acquiring data, you must specify in the **trigger slope** input if the acquisition is going to be triggered on the rising or falling edge of the analog trigger signal. Aside from specifying the slope, you must enter the

**trigger channel** where the analog triggering signal will be connected as well as the **trigger level** on the triggering signal needed to begin acquisition. Once you specify the channel of the triggering signal, LabVIEW waits until the slope and trigger level conditions are met before starting a buffered acquisition. If you use channel names configured in the DAQ Channel Wizard, **trigger level** is treated as being relative to the physical units specified for the channel in the DAQ Channel Wizard. Otherwise, **trigger level** is treated as volts.

The Acquire N Scans Analog Hardware Trig VI example, located in `labview\examples\daq\anlogin\anlogin.llb`, holds the data in a memory buffer until the device completes data acquisition. The number of data points you want to acquire must be small enough to fit in memory. This VI only views and processes the information after the acquisition. If you need to view and process information during the acquisition, use the Acquire & Proc N Scans-Trig VI, located in `labview\examples\daq\anlogin\anlogin.llb`. If you expect multiple analog trigger signals that will start multiple acquisitions, use the example Acquire N-Multi-Analog Hardware Trig VI, located in `labview\examples\daq\anlogin\anlogin.llb`.

# Software Triggering

With software triggering, you can simulate an analog trigger using software. This form of triggering is often used in situations where hardware triggers are not available. Another name for software triggering signals, specifically analog signals, is *conditional retrieval*. With conditional retrieval, you set up your DAQ device to collect data, but the device does not return any data to LabVIEW unless the data meets your retrieval conditions. LabVIEW scans the input data and performs a comparison with the conditions, but does not store the data until it meets your specifications. Figure 8-8 shows a timeline of events that typically occur when you perform conditional retrieval.

The read/search position pointer traverses the buffer until it finds the scan location where the data has met the retrieval conditions. Offset indicates the scan location from which the VI begins reading data relative to the read/search position. A negative offset indicates that you need pretrigger data (data prior to the retrieval conditions). If offset is greater than 0, you need posttrigger data (data after retrieval conditions).

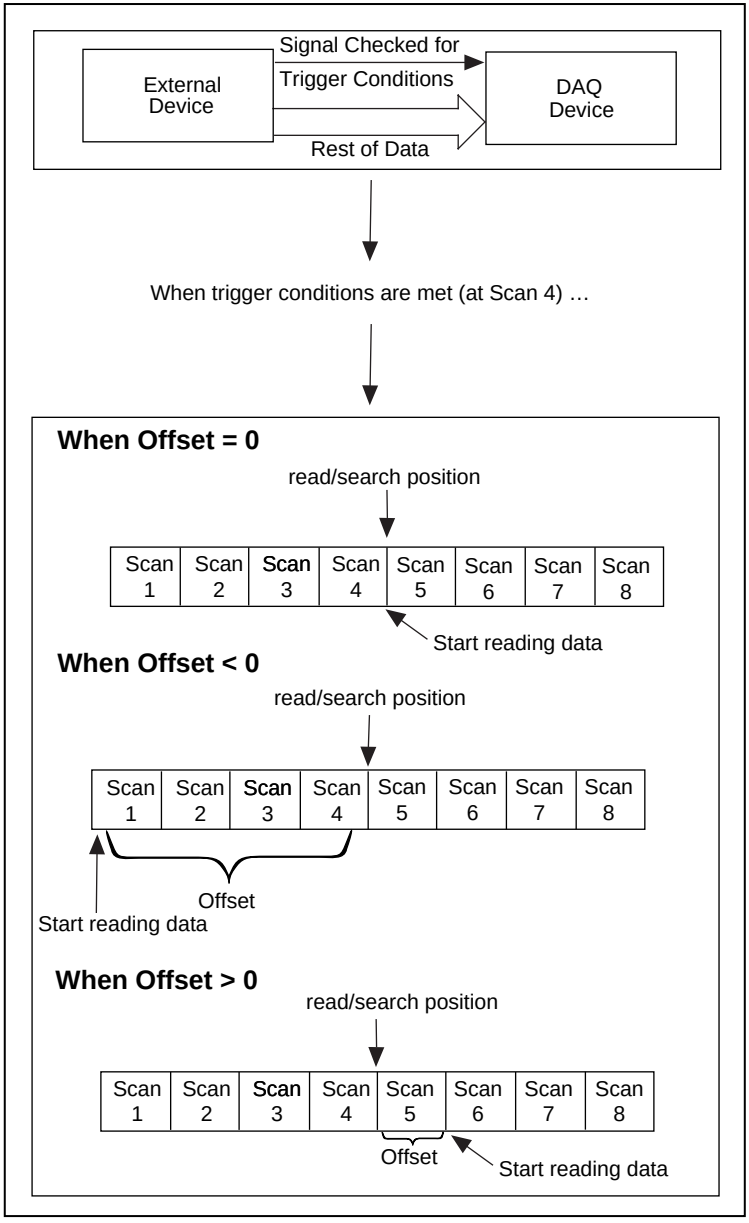



**Figure 8-8.** Timeline of Conditional Retrieval

The **conditional retrieval** cluster of the AI Read VI specifies the analog signal conditions of retrieval, as shown in Figure 8-9.

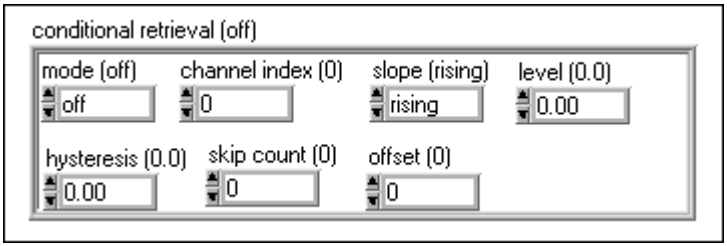



**Figure 8-9.** The AI Read VI Conditional Retrieval Cluster

**Tip** The actual data acquisition is started by running your VI and the conditional retrieval just controls how data already being acquired is returned.

When acquiring data with conditional retrieval, you typically store the data in a memory buffer, similar to hardware triggering applications. After you start running the VI, the data is placed in the buffer. Once the retrieval conditions have been met, the AI Read VI searches the buffer for the desired information. As with hardware analog triggering, you specify the analog channel of the triggering signal by specifying its **channel index,** an index number corresponding to the relative order of a single channel in a channel list. You also specify the **slope** (rising or falling) and the **level** of the trigger signal.

**Note** The channel index might not be equal to the channel value. You can use the Channel to Index VI to get the channel index for a channel. You can find this VI in **Data Acquisition»Calibration and Configuration**.

The AI Read VI begins searching for the retrieval conditions in the buffer at the read/search position, another input of the AI Read VI. The **offset**, a value of the **conditional retrieval** input cluster, is where you specify the scan locations from which the VI begins reading data relative to the read/search position. A negative **offset** indicates data prior to the retrieval condition pretrigger data, and a positive **offset** indicates data after the retrieval condition posttrigger data. The **skip count** input is where you specify the number of times the trigger conditions are met. The **hysteresis** input is where you specify the range you will use to meet retrieval conditions. Once the **slope** and **level** conditions on **channel index** have been found, the read/search position indicates the location where the retrieval conditions were met.

If you are using channel names configured in the DAQ Channel Wizard, **level** and **hysteresis** are treated as being relative to the physical units specified for the channel. If you are not using channel names, these inputs are treated as volts.

For more information on the **conditional retrieval** input cluster, look at the AI Read VI description in Chapter 16, *Intermediate Analog Input VIs*, in the *LabVIEW Function and VI Reference Manual*, or the LabVIEW *Online Reference*, available by selecting **Help»Online Reference**.

## Conditional Retrieval Examples

The Acquire N Scans Analog Software Trig VI example, located in `labview\examples\daq\anlogin\anlogin.llb`, uses the Intermediate VIs, as shown in Figure 8-10.

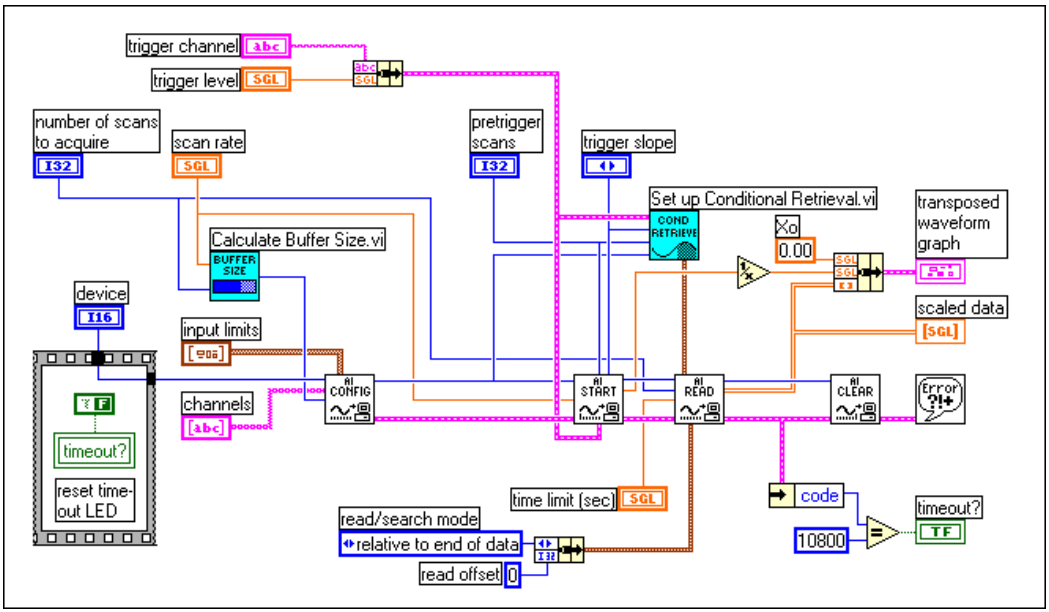



**Figure 8-10.** Block Diagram of the Acquire N Scans Analog Software Trig VI

The main difference between this software triggering example and hardware triggering is the use of the **conditional retrieval** input for the AI Read VI. You set up the **trigger channel**, **trigger slope**, and **trigger level** the same way for both triggering methods. The **pretrigger scans** value is negated and connected to the **offset** value in the **conditional retrieval** cluster of the AI Read VI. When the trigger conditions are met, the VI returns the requested number of scans.

# 9

# Letting an Outside Source Control Your Acquisition Rate

Typically, a DAQ device uses internal counters to determine the rate to acquire data, but sometimes you might need to capture your data at the rate of particular signals in your system. For example, you can also read temperature channels every time a pulse occurs, which represents pressure rising above a certain level. In this case, internal counters are inefficient for your needs. You must control your acquisition rate by some other, external source.

You can compare a scan of your channels to taking a snapshot of the voltages on your analog input channels. If you set your scan rate to 10 scans per second, you are taking 10 snapshots each second of all the channels in your channel list. In this case, an internal clock within your device (the scan clock) sets the scan rate, which controls the time interval between scans.

Also, remember that most DAQ devices (those that do not sample simultaneously) proceed from one channel to the next or from one sample to the next, depending on the channel clock rate. Therefore, the channel clock is the clock controlling the time interval between individual channel samples within a scan, which means the channel clock proceeds at a faster rate than the scan clock.

The faster the channel clock rate, the more closely in time your system samples the channels within each scan, as shown in Figure 9-1.

**Note** For devices with both a scan and channel clock, lowering the scan rate does not change the channel clock rate.

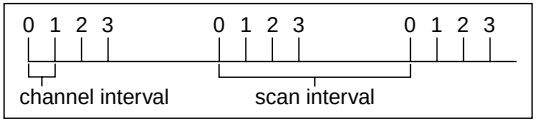



**Figure 9-1.** Channel and Scan Intervals Using the Channel Clock

Some DAQ devices do not have scan clocks, but rather use *round-robin scanning*. Figure 9-2 shows an example of round-robin scanning.

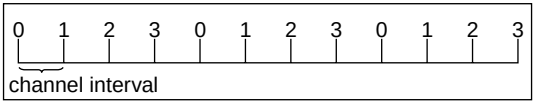



**Figure 9-2.** Round-Robin Scanning Using the Channel Clock

The devices that always perform round-robin scanning include, but are not limited to, the following:

- NB-MIO-16
- PC-LPM-16
- PC-LPM-16PnP
- PC-516
- DAQCard-500
- DAQCard-516
- DAQCard-700
- Lab-NB, Lab-SE
- Lab-LC
- NI 4060

With no scan clock, the channel clock is used to switch between each channel at an equal interval. The same delay exists among all channel samples, as well as between the last channel of a scan and the first channel in the next scan. For boards with scan and channel clocks, round-robin scanning occurs when you disable the scan clock by setting the scan rate to `0` and using the **interchannel delay** of the AI Config VI to control your acquisition rate.

LabVIEW is *scan-clock oriented*. In other words, when you select a scan rate, LabVIEW automatically selects the channel clock rate for you. LabVIEW selects the fastest channel clock rate that allows adequate settling time for the ADC.

LabVIEW adds an extra 10 µs to the interchannel delay to compensate for any unaccounted factors. However, LabVIEW does not consider this additional delay for purposes of warnings. If you have specified a scan rate that is adequate for acquisition but too fast for LabVIEW to apply the 10-µs delay, it configures the acquisition but does not return a warning.

You can set your channel clock rate with the **interchannel delay** input of the AI Config VI, which calls the Advanced AI Clock Config VI to actually configure the channel clock. The simplest method to select an interchannel delay is to gradually increase the delay, or clock period, until the data appears consistent with data from the previous delay setting.

Refer to your hardware manuals for the required settling time for your channel clock. You can also find the interchannel delay by running the low-level AI Clock Config VI for the channel clock with no frequency specified.

# Externally Controlling Your Channel Clock

There are times when you might need to control the channel clock externally. The channel clock rate is the same rate at which analog conversions occur. For instance, suppose you need to know the strain value at an input, every time an infrared sensor sends a pulse. Most DAQ devices have an EXTCONV* pin or a PFI pin on the I/O connector for providing your own channel clock. For NI 406*x* Series devices, use the EXTRIG input pin. This external signal must be a TTL level signal. The asterisk on the signal name indicates that the actual conversion occurs on the falling edge of the signal, as shown in Figure 9-3. For devices with PFI lines and for the NI 406*x* Series devices, you can select either the rising edge or falling edge using LabVIEW. With devices that have a RTSI connector, you can get your channel clock from other National Instruments DAQ devices.

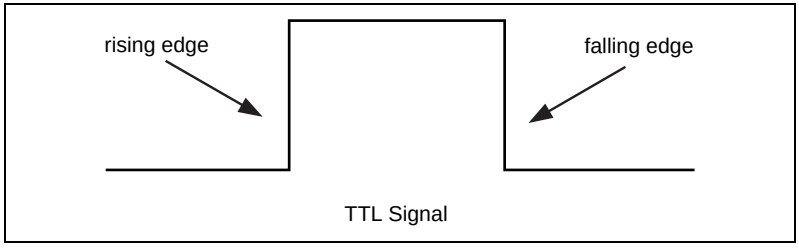



**Figure 9-3.** Example of a TTL Signal

Figure 9-4 shows the Acquire N Scans-ExtChanClk VI, located in `labview\examples\daq\anlogin\anlogin.llb`. This example demonstrates how to set up your acquisition for an externally controlled channel clock. The VI includes the AI Clock Config VI and the clock source was connected to the I/O connector.

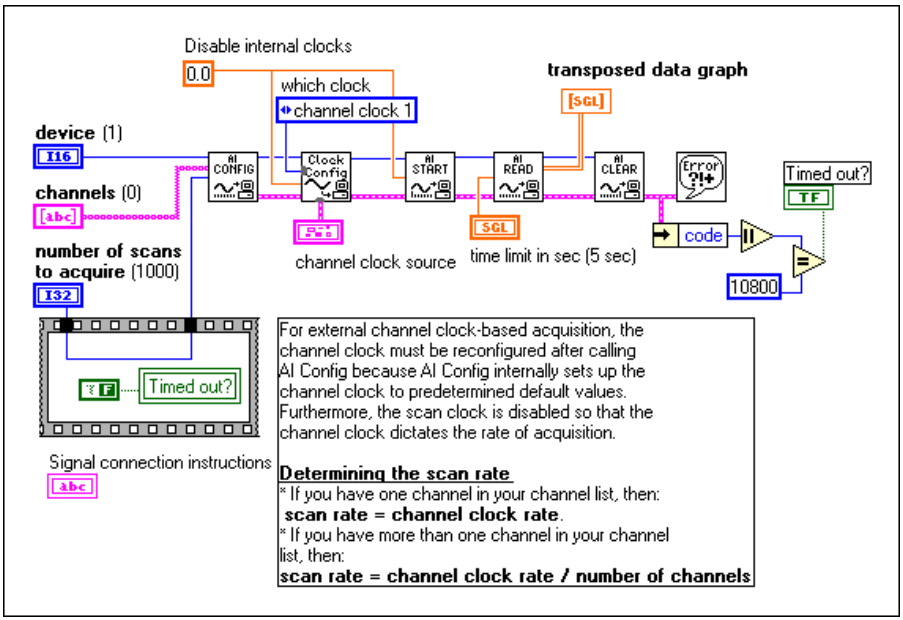

Disable internal clocks
0.0
which clock
channel clock 1
transposed data graph
[SGL]
device (1)
I16
channels (0)
[abc]
number of scans to acquire (1000)
I32
AI CONFIG
Clock Config
AI START
AI READ
AI CLEAR
Error
Timed out?
TF
code
10800
channel clock source
SGL
time limit in sec (5 sec)
Timed out?
Signal connection instructions
abc
For external channel clock-based acquisition, the channel clock must be reconfigured after calling AI Config because AI Config internally sets up the channel clock to predetermined default values. Furthermore, the scan clock is disabled so that the channel clock dictates the rate of acquisition.
Determining the scan rate
* If you have one channel in your channel list, then:
scan rate = channel clock rate.
* If you have more than one channel in your channel list, then:
scan rate = channel clock rate / number of channels

**Figure 9-4.** Getting Started Analog Input Example VI

You can enable external conversions by calling the advanced-level AI Clock Config VI. Remember that the AI Clock Config VI, which is called by the AI Config VI, normally sets internal channel delay automatically or manually with the **interchannel delay** control. However, calling the AI Clock Config VI after the AI Config VI resets the channel clock so that it comes from an external source for external conversion. Also, notice that the scan clock is set to 0 to disable it, allowing the channel clock to control the acquisition rate.

**Note** The 5102 devices do not support external channel clock pulses, because there is no channel clock on the device.

On most devices, external conversions occur on the falling edge of the EXTCONV* line. Consult your hardware reference manual for timing diagrams. On devices with PFI lines (such as E Series devices), you can set the **Clock Source Code** input of AI Clock Config VI to the PFI pin with either falling or rising edge or use the default PFI2/Convert* pin where the conversions occur on the falling edge, as shown in Figure 9-5.

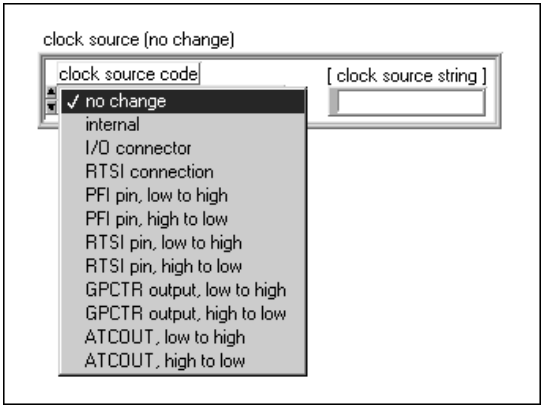



**Figure 9-5.** Setting the Clock Source Code for External Conversion Pulses for E Series Devices

**Note** The AT-MIO-16, AT-MIO-16D, NB-MIO-16, and NB-MIO-16X cannot support both an external channel clock and a digital trigger signal at the same time. You must choose one or the other.

Because LabVIEW determines the length of time before the AI Read VI times out based on the **interchannel delay** and **scan clock rate**, you may need to force a time limit for the AI Read VI, as shown previously in Figure 9-4.

**Note** On the Lab-PC+ and 1200 devices, the first clock pulse on the EXTCONV* pin configures the acquisition but does not cause a conversion. However, all subsequent pulses cause conversions.

Figure 9-6 shows an example of using an external scan clock to perform a buffered acquisition.

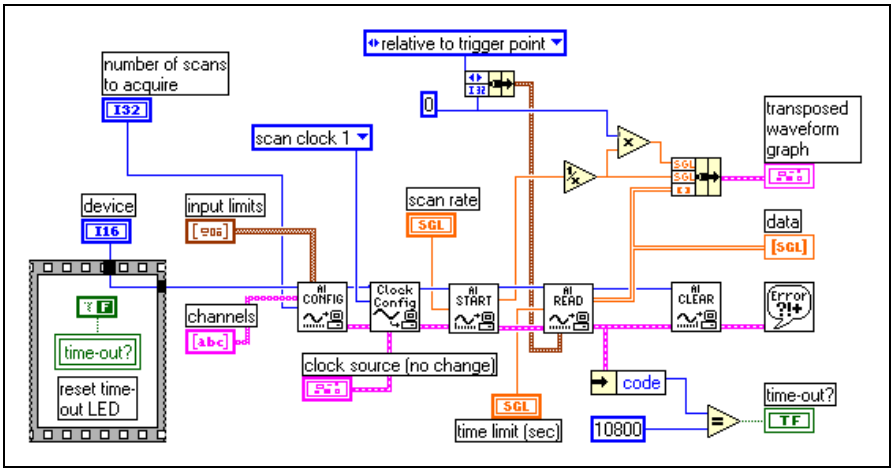



**Figure 9-6.** Block Diagram of a VI Acquiring Data On with an External Scan Clock

# Externally Controlling Your Scan Clock

External scan clock control might be more useful than external channel clock control if you are sampling multiple channels, but might not be as obvious to find because it does not have the input on the I/O connector labeled `ExtScanClock`, the way the EXTCONV* pin does.

**Note** Some MIO devices have an output on the I/O connector labeled `SCANCLK`, which is used for external multiplexing and is not the analog input scan clock. This cannot be used as an input.

The appropriate pin to input your external scan clock can be found in Table 9-1.

**Table 9-1.** External Scan Clock Input Pins

| Device | External Scan Clock Input Pin |
|---|---|
| All E Series Devices | Any PFI Pin<br>(Default: PF17/STARTSCAN) |
| Lab-PC+<br>1200 devices | OUT B1 |
| AT-MIO-16<br>AT-MIO-16F-5<br>AT-MIO-16X<br>AT-MIO-16D<br>AT-MIO-64F-5 | OUT2 |

**Note** Some devices do not have internal scan clocks and therefore do not support external scan clocks. These devices include, but are not limited to the following: NB-MIO-16, PC-LPM-16, PC-LPM-16PnP, PC-516, DAQCard-500, DAQCard-516, DAQCard-700, NI 4060, Lab-NB, Lab-SE, and Lab-LC.

After connecting your external scan clock to the correct pin, set up the external scan clock in software. In Figure 9-7, the example Acquire N Scans-ExtScanClk VI located in `labview\examples\daq\anlogin\anlogin.llb` shows how to do this. Two Advanced VIs, AI Clock Config and AI Control, are used in place of the Intermediate AI Start VI. This allows access to the **clock source** input. This is necessary because it allows access to the **clock source string**, which is used to identify the PFI pin to be used for the scan clock for E Series boards. The **clock source** also includes the **clock source code** (on the front panel), which is set to I/O connector. The `0.0` wired to the Clock Config VI disables the internal clock.

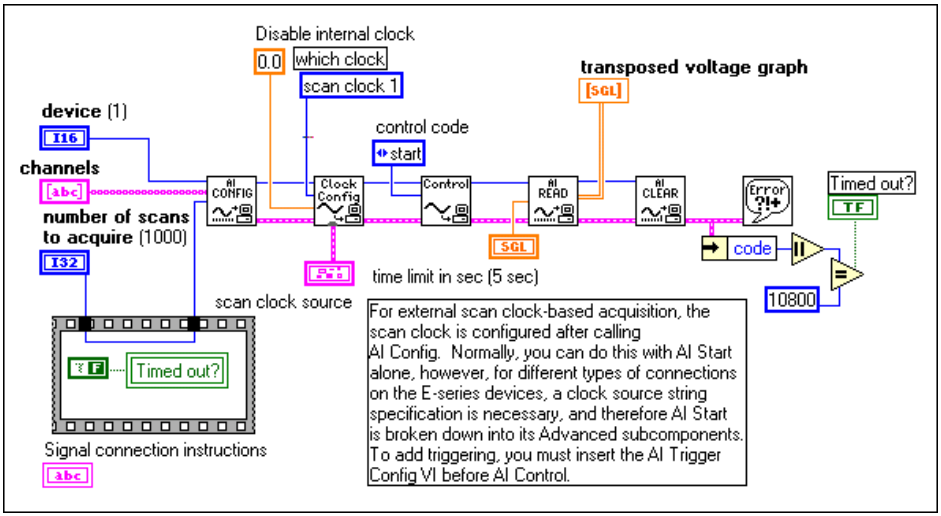
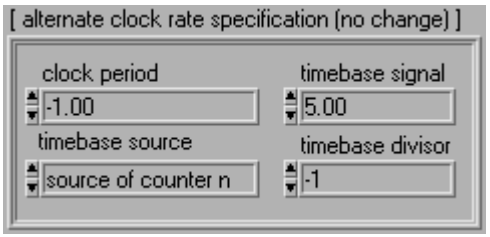

Disable internal clock
0.0
which clock
scan clock 1
transposed voltage graph
SGL
device (1)
I16
control code
start
channels
abc
AI CONFIG
Clock Config
Control
AI READ
AI CLEAR
Error
Timed out?
TF
number of scans to acquire (1000)
I32
SGL
code
10800
time limit in sec (5 sec)
scan clock source
For external scan clock-based acquisition, the scan clock is configured after calling AI Config. Normally, you can do this with AI Start alone, however, for different types of connections on the E-series devices, a clock source string specification is necessary, and therefore AI Start is broken down into its Advanced subcomponents. To add triggering, you must insert the AI Trigger Config VI before AI Control.
Timed out?
Signal connection instructions
abc

**Figure 9-7.** Externally Controlling Your Scan Clock with the Getting Started Analog Input Example VI

The NB-MIO-16X cannot support external scan clocks as the other devices can. The device layout does not allow you to directly provide an external scan clock. Instead, you can offer a timebase to the internal counter, counter 5, that generates the scan clock. Do this by sending a timebase into the SOURCE5 pin and calling the Advanced VI, AI Clock Config. In addition, wire the **alternate clock rate specification,** as shown in the following figure, into the AI Clock Config VI.




Remember that the **which clock** input of the AI Clock Config VI should be set to `scan clock (1)`.

**Note** You must divide the timebase by some number between 2 and 65,535 or you will get a bad input value error.

Because LabVIEW determines the length of time before AI Read times out based on the interchannel delay and scan clock rate, you may need to force a time limit into AI Read. In Figure 9-7, the time limit is 5 seconds.

# Externally Controlling the Scan and Channel Clocks

You can control the scan and channel clocks simultaneously. However, make sure that you follow the proper timing. Figure 9-8 demonstrates how you can set up your application to control both clocks.

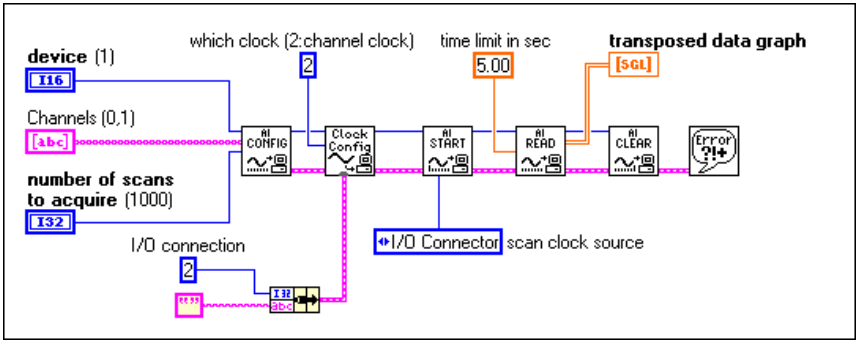



**Figure 9-8.** Controlling the Scan and Channel Clock Simultaneously

# Part III

# Making Waves with Analog Output

This section contains basic information about generating data with LabVIEW, including generating a single point or multiple points.

Part III, *Making Waves with Analog Output*, contains the following chapters:

- Chapter 10, *Things You Should Know about Analog Output*, explains how to use LabVIEW to produce all of the different types of analog output signals.
- Chapter 11, *One-Stop Single-Point Generation*, shows you which VIs to use in LabVIEW to perform single-point updates.
- Chapter 12, *Buffering Your Way through Waveform Generation*, shows you which VIs to use in LabVIEW to perform buffered analog updates.
- Chapter 13, *Letting an Outside Source Control Your Update Rate*, shows you which VIs to use in LabVIEW to control your update rate with an external source.
- Chapter 14, *Simultaneous Buffered Waveform Acquisition and Generation*, describes how to perform buffered waveform acquisition and generation simultaneously on the same DAQ device.

# 10

# Things You Should Know about Analog Output

Some measuring systems require that analog signals be generated by a DAQ device. Each of these analog signals can be a steady or slowly changing signal, or a continuously changing waveform. This chapter describes how to use LabVIEW to produce all of these different types of signals.

## Single-Point Output

When the signal level at the output is more important than the rate at which the output value changes, you need to generate a steady DC value. You can use the single-point analog output VIs to produce this type of output. With single-point analog output, any time you want to change the value on an analog output channel, you must call one of the VIs that produces a single update (a single value change). Therefore, you can change the output value only as fast as LabVIEW calls the VIs. This technique is called *software timing*. You should use software timing if you do not need high-speed generation or very accurate timing. Refer to Chapter 11, *One-Stop Single-Point Generation*, for more information on single-point output.

## Buffered Analog Output

Sometimes in performing analog output, the rate that your updates occur is just as important as the signal level. This is called *waveform generation*, or *buffered analog output*. For example, you might want your DAQ device to act as a function generator. You can do this by storing one cycle of sine wave data in an array and programming the DAQ device to generate the values continuously in the array one point at a time at a specified rate. This is known as *single-buffered waveform generation*. But what if you want to generate a continually changing waveform? For example, you might have a large file stored on disk that contains data you want to output. Because LabVIEW cannot store the entire waveform in a single buffer, you must

continually load new data into the buffer during the generation. This process requires the use of *circular-buffered* analog output in LabVIEW. To learn more about single or circular buffering, read Chapter 12, *Buffering Your Way through Waveform Generation*.

# 11

# One-Stop Single-Point Generation

This chapter shows you which VIs to use in LabVIEW to perform single-point updates.

## Single-Immediate Updates

The simplest way to program single-point updates in LabVIEW is by using the Easy Analog Output VI, AO Update Channels. Figure 11-1 shows a diagram of a VI that writes values to one or more output channels on the output DAQ device.

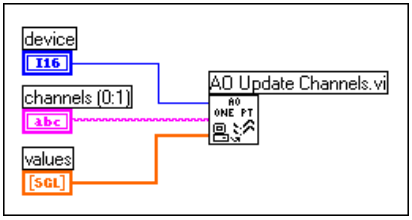



**Figure 11-1.** Single Immediate Update Using the AO Update Channels VI

Notice that an array of values is passed as an input to the VI. The first element in the array corresponds to the first entry in the channel string, and the second array element corresponds to the second channel entry. If you use channel names configured in the DAQ Channel Wizard in your channel string, **values** is relative to the physical units you specify in the DAQ Channel Wizard. Otherwise, **values** is relative to volts. For more information on channel string syntax, refer to Chapter 3, *Basic LabVIEW Data Acquisition Concepts*. Remember that Easy VIs already have built-in error handling.

While Figure 11-1 shows how to write values for multiple channels, Figure 11-2 shows the diagram of the Generate 1 Point on 1 Channel VI located in `labview\examples\daq\anlogout\anlogout.llb`, which generates one value for one channel.

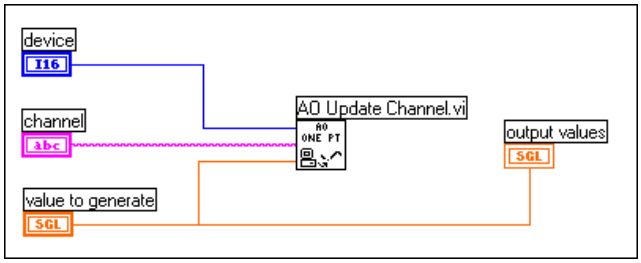



**Figure 11-2.** Single Immediate Update Using the AO Update Channel VI

If you want more control over the limit settings for each channel, you also can program a single-point update using the Intermediate Analog Output VI, AO Write One Update. Figure 11-3 shows an example of using this VI.

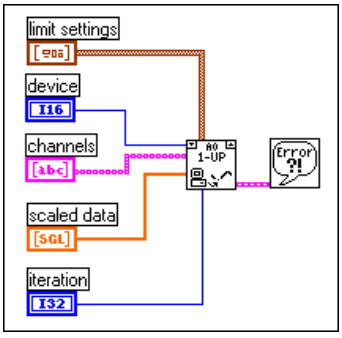



**Figure 11-3.** Single Immediate Update Using Intermediate VI

In this example, your program passes the error information to the Simple Error Handler VI. The **iteration** input optimizes the execution of this VI if you place it in a loop. With Intermediate VIs, you gain more control over when you can check for errors.

# Multiple-Immediate Updates

Figure 11-4 shows the block diagram of a VI that performs multiple updates. The Write N Updates example VI, located in `labview\examples\daq\anlogout\anlogout.llb`, is similar to Figure 11-4. The diagram shown in Figure 11-4 resembles the one shown in Figure 11-3, except that the While Loop executes the subVI repeatedly until either the error status or the stop Boolean is TRUE. You can use the Easy Analog Output VI, AO Write One Update, in a loop, but this is inefficient because the Easy I/O VIs configure the device every time they execute. The AO Write One Update VI configures the device only when the value of the **iteration** input is set to `0`.

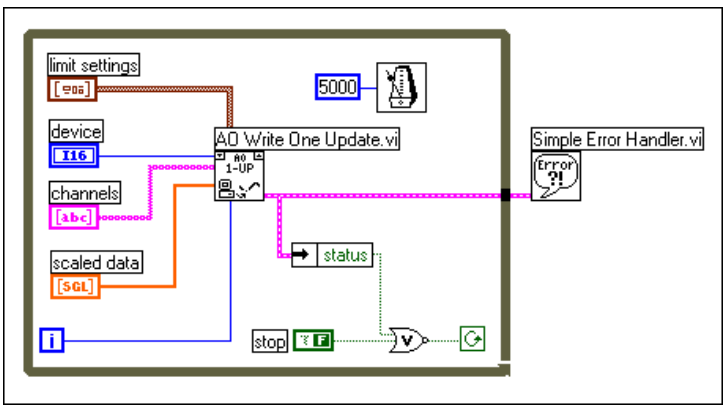



**Figure 11-4.** Multiple Immediate Updates Using Intermediate VI

Figure 11-4 shows an immediate, software-timed analog output VI application. This means that software timing in a loop controls the update rate. One good reason to use immediate, software-timed output is that your application calculates or processes output values one at a time. However, remember that software timing is not as accurate as hardware-timed analog output. For more information on hardware-timed analog output, refer to Chapter 6, *One-Stop Single-Point Acquisition*.

# 12

# Buffering Your Way through Waveform Generation

This chapter shows you which VIs to use in LabVIEW to perform buffered analog updates.

## Buffered Analog Output

You can program single-buffered analog output in LabVIEW using an Easy Analog Output VI, AO Generate Waveforms VI, as shown in Figure 12-1. This VI writes an array of output values to the analog output channels at a rate specified by **update rate**. For example, if **channels** consists of two channels and the **waveforms** 2D array consists of two columns containing data for the two channels, LabVIEW writes values from each column to the corresponding channels at every update interval. After LabVIEW writes all the values in the 2D array to the channels, the VI stops. The signal level on the output channels maintains the value of the final value row in the 2D array until another value is generated. If you use channel names configured in the DAQ Channel Wizard in **channels**, **waveforms** is relative to the units specified in the DAQ Channel Wizard. Otherwise, **waveforms** is relative to volts.

Easy VIs contain error handling. If an error occurs in the AO Generate Waveforms VI, a dialog box appears displaying the error number and description, and the VI stops running.

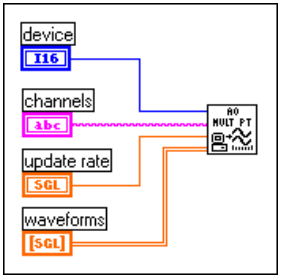



**Figure 12-1.** Waveform Generation Using the AO Generate Waveforms VI

As with single-point analog output, you can use the Analog Output Utility VI, AO Waveform Gen VI, for most of your programming needs. This VI has several inputs and outputs that the Easy I/O VI does not have. You have the option of having the data array generated once, several times, or continuously through the **generation count** input. Figure 12-2 shows an example diagram of how to program this VI.

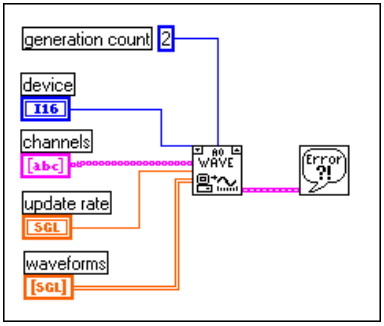



**Figure 12-2.** Waveform Generation Using the AO Waveform Gen VI

In this example, LabVIEW generates the data in the array two times before stopping.

The Generate N Updates example VI, located in `labview\examples\daq\anlogout\anlogout.llb`, uses the AO Waveform Gen VI. Placing this VI in a loop and wiring the iteration terminal of the loop to the iteration input on the VI optimizes the execution of this VI. When iteration is 0, LabVIEW configures the analog output channels appropriately. If the iteration is greater than 0, LabVIEW uses the existing configuration, which improves performance. With the AO Waveform Gen VI, you also can specify the limit settings input for each analog output channel. For more information on limit settings, refer to Chapter 3, *Basic LabVIEW Data Acquisition Concepts*.

If you want even more control over your analog output application, use the Intermediate DAQ VIs, as shown in Figure 12-3.

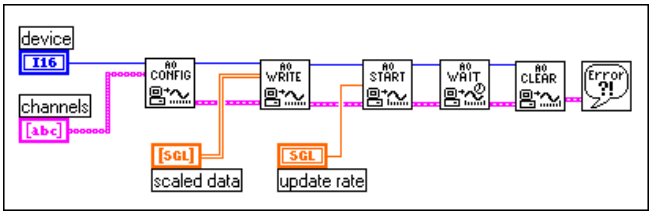



**Figure 12-3.** Waveform Generation Using Intermediate VIs

With these VIs, you can set up an alternate update clock source (such as an external clock or a clock signal coming from another device) or return the update rate. The AO Config VI sets up the channels you specify for analog output. The AO Write VI places the data in the buffer, the AO Start VI begins the actual generation at the **update rate**, and the AO Wait VI waits until the waveform generation completes. Then, the AO Clear VI clears the analog channels.

The Generate Continuous Sinewave VI, located in `labview\examples\daq\anlogout\anlogout.llb`, is similar in structure to Figure 12-3. This example VI continually outputs a sine waveform through the channel you specify.

# Changing the Waveform during Generation—Circular-Buffered Output

When the waveform data is too large to fit in a memory buffer or is constantly changing, use a *circular buffer* to output the data. You also can use the Easy Analog Output VIs in a loop to create a circular-buffered output; but this sacrifices efficiency because Easy VIs configure, allocate, and deallocate a buffer every time they execute, which causes time gaps between the data output. Figures 12-4 and 12-5 show two different ways to perform circular-buffered analog output using the Intermediate VIs in

LabVIEW. Figure 12-4 shows the AO Continuous Gen VI, which is more efficient than the Easy Analog Output VIs in that it configures and allocates a buffer when its **iteration** input is 0 and deallocates the buffer when the **clear generation** input is TRUE.

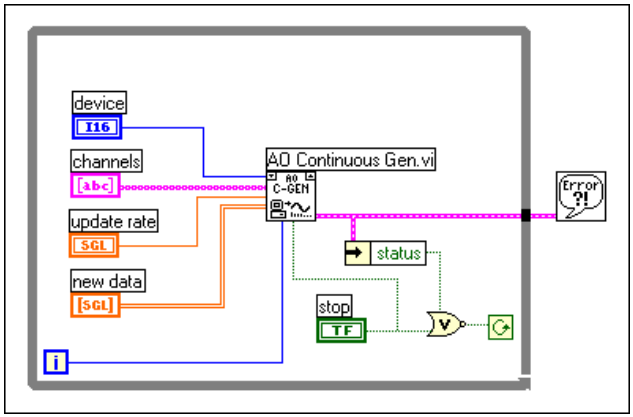



**Figure 12-4.** Circular Buffered Waveform Generation Using the AO Continuous Gen VI

With the AO Continuous Gen VI, you can configure the size of the data buffer and the limit settings of each channel. For more information on how to set limit settings, refer to Chapter 3, *Basic LabVIEW Data Acquisition Concepts*.

The Continuous Generation example VI, located in `labview\examples\daq\anlogout\anlogout.llb`, uses the AO Continuous Gen VI. In this example, the data completely fills the buffer on the first iteration. On subsequent iterations, new data is written into one half of the buffer while the other half continues to output data.

To gain more control over your analog output application, use the Intermediate VIs shown in Figure 12-5. With these VIs, you can set up an alternate update clock source and you can monitor the update rate the VI actually uses. The AO Config VI sets up the channels you specify for

analog output. The AO Write VI places the data in a buffer. The AO Start VI begins the actual generation at the **update rate**. The AO Write VI in the While Loop writes new data to the buffer until you click the **Stop** button. Then, the AO Clear VI clears the analog channels.

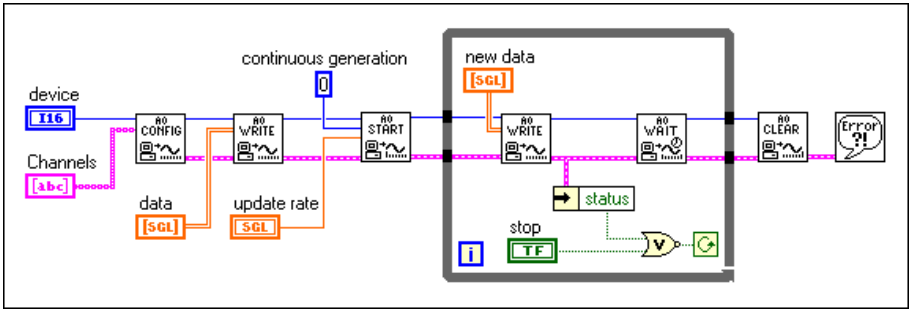



**Figure 12-5.** Circular Buffered Waveform Generation Using Intermediate VIs

The Function Generator VI, located in `labview\examples\daq\anlogout\anlogout.llb`, is a more advanced example than the one shown in Figure 12-5. This VI changes the output waveform on-the-fly, responding to changing signal types (sine or square), amplitude, offset, update rate, and phase settings on the front panel.

# Eliminating Errors from Your Circular-Buffered Application

If you get error number **`-10843 underFlowError`**, while performing circular-buffered output, it means your program cannot write data fast enough to the buffer to output the data at the update rate. To solve this problem, decrease the speed of the update rate. If adjusting the update rate does not get rid of the error in your application, increase the buffer size.

# Buffered Analog Output Examples

Another example VI in this library you might find helpful, Display and Output Acq'd File (scaled) VI, is shown in Figure 12-6.

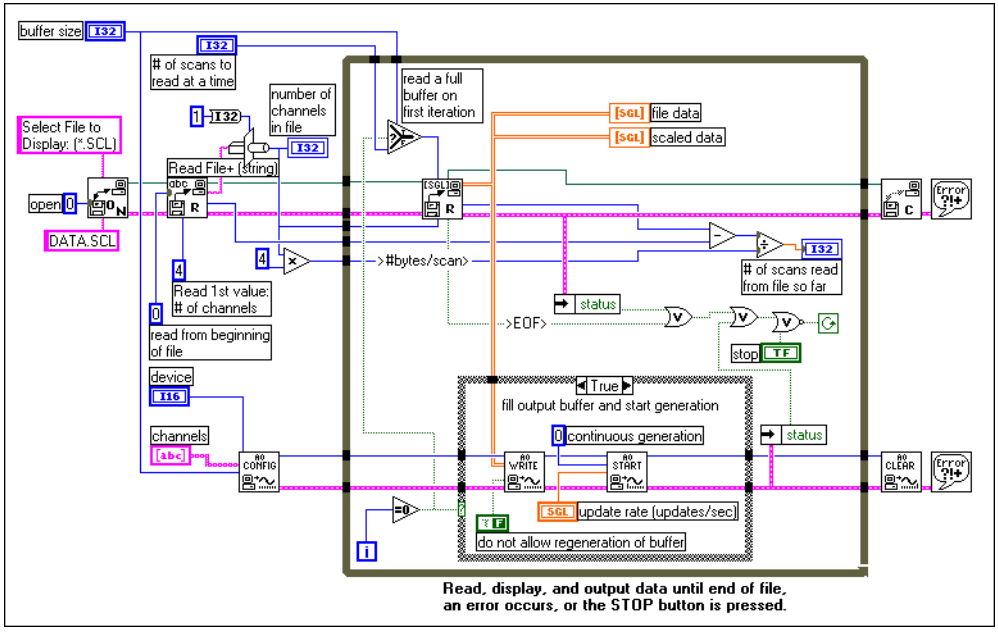



**Figure 12-6.** Display and Output Acq'd File (Scaled) VI

You can use this VI in conjunction with the Cont Acq to File (scaled) VI, located in `labview\examples\daq\anlogin\anolgin.llb`. The Display and Output Acq'd File (scaled) VI also is described in Chapter 7, *Buffering Your Way through Waveform Acquisition*. After running the Cont Acq to File (scaled) VI and saving your acquired data to disk, you can run the Display and Output Acq'd File (scaled) VI to generate your data from the file you created. This example uses circular-buffered output. To generate data at the same rate at which it was acquired, you must know the rate at which your data was acquired, and use that as the **update rate**.

# 13

# Letting an Outside Source Control Your Update Rate

DAQ devices use internal counters and timers to determine the rate of data generation. However, you might encounter times when you need to generate data in synch with other signals in your system. For example, you might need to output data to a test circuit every time that test circuit emits a pulse. In this case, internal counter/timers are inefficient for your needs. You need to control the update rate with your own external source of pulses.

## Externally Controlling Your Update Clock

Chapter 12, *Buffering Your Way through Waveform Generation*, mentions that for more control over your analog output applications, you can use the Intermediate DAQ VIs. This chapter explains how to use these Intermediate VIs to generate data using an external update clock.

The update clock controls the rate digital to analog conversions occur. To control your data generation externally, you must supply this clock signal to the appropriate pin on the I/O connector of your DAQ device. The clock source you supply must be a TTL signal. Figure 13-1 shows the Generate N Updates-ExtUpdateClk VI, located in `labview\examples\daq\anlogout\anlogout.llb`, which applies this process.

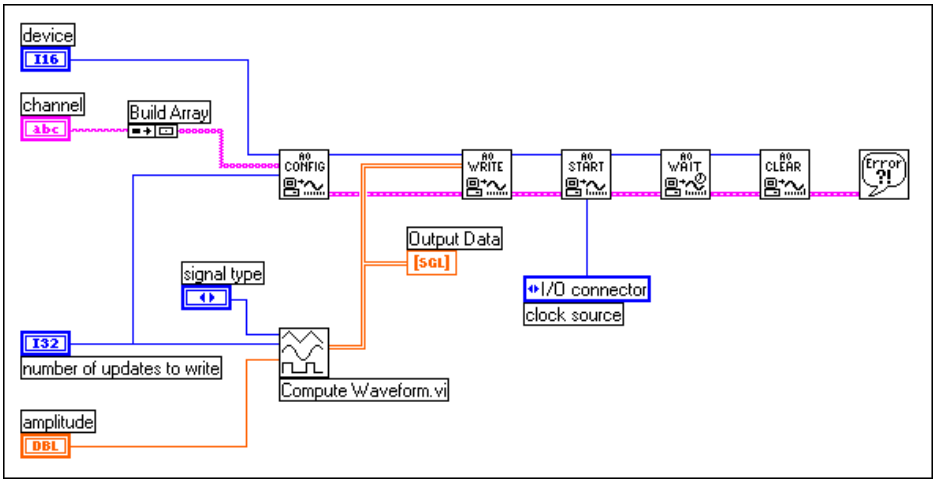



**Figure 13-1.** Generate N Updates-ExtUpdateClk VI

To use an external update clock, you must set the **clock source** of the AO Start VI to `I/O connector`. When you connect your external clock, you find that different DAQ devices use different pins for this input. However, if you select **Show VI Info** in the **Windows** menu of the example VI, you find that all the I/O connections are explained for you. These input pins also are described in Table 13-1.

**Table 13-1.** External Update Clock Input Pins

| Device | External Update Clock Input Pin |
|---|---|
| All E Series Devices with analog output | Any PFI pin (Default: PFI5/UPDATE*) |
| Non E Series MIO type devices | OUT2 |
| Lab-PC+ 1200 devices AT-AO-6/10 | EXTUPDATE* |

For waveform generation, you must supply an array of waveform data. The example VI in Figure 13-1 uses data created in the Compute Waveform VI. When you run the example VI, the data is output on channel 0 (the DAC0OUT pin) of your DAQ device.

# Supplying an External Test Clock from Your DAQ Device

To use an external update clock when you do not have an external clock available, create an external test clock using outputs from a counter/timer on your DAQ device, and then wire the output to your external update clock source.

If your DAQ device has an FOUT or FREQ_OUT pin, you can generate a 50% duty cycle TTL pulse train using the Generate Pulse Train on FOUT or FREQ_OUT VI, located in `labview\examples\daq\counter\DAQ-STC.llb`. The advantage of this VI is that it does not use one of the available counters, which you might need for other reasons.

You can also use the Pulse Train VIs to create an external test clock. These VIs are located in `examples\daq\counter\DAQ-STC.llb`, `examples\daq\counter\MI-TIO.llb`, `examples\daq\counter\Am9513.llb`, and `examples\daq\counter\8253.llb`.

# 14

# Simultaneous Buffered Waveform Acquisition and Generation

This chapter describes how to perform buffered waveform acquisition and generation simultaneously on the same DAQ device.

## Using E Series MIO Boards

E series devices, such as the PCI-MIO-16E-1, have separate counters dedicated to analog input and analog output timing. For this reason, they are the best choice for simultaneous input/output.

### Software Triggered

Figure 14-1 shows the block diagram of the Simul AI/AO Buffered (E Series MIO) VI located in `labview\examples\daq\anlog_io\anlog_io.llb`.

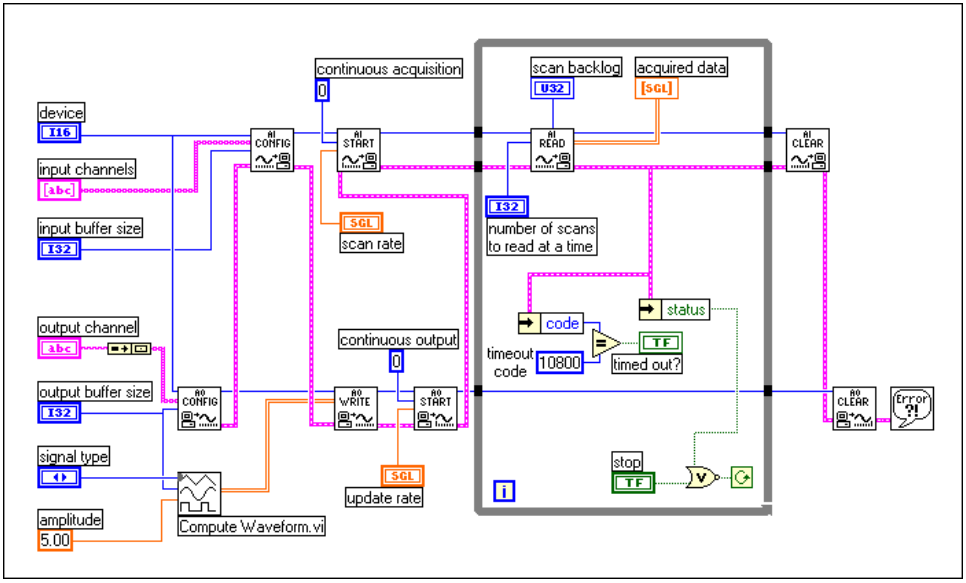



**Figure 14-1.** Simultaneous Input/Output Using the Simul AI/AO Buffered (E-series MIO) VI

This example VI uses Intermediate DAQ VIs. This example VI uses the same VIs you used for analog input in Chapter 7, *Simultaneous Buffered Waveform Acquisition and Generation*—AI Config, AI Start, AI Read, and AI Clear—for waveform acquisition. This example VI also uses the same VIs you used for analog output in Chapter 12, *Buffering Your Way through Waveform Generation*—AO Config, AO Write, AO Start, and AO Clear—for waveform generation here. By following the **error** cluster wire, which enters each DAQ VI on the bottom left and exits on the bottom right, you can see that because of data dependency, the waveform generation starts before the waveform acquisition, and each task is configured to run continuously. This example VI is software triggered because it starts via software when you click the **Run** button.

Once you call the AO Start and AI Start VIs, the While Loop executes. Inside the While Loop, the AI Read VI returns acquired data from the analog input buffer. There is not a call to the AO Write VI inside the While Loop because it is not needed if the same data from the first AO Write VI is regenerated continuously. To generate new data each time the While Loop iterates, add an AO Write VI inside the While Loop. The While Loop stops when an error occurs or you click the **Stop** button. Your DAQ device resources are cleared by calling the AI Clear and AO Clear VIs after the loop stops.

For a complete description, instructions, and I/O connections for this VI, select **Windows»Show VI Info** from the front panel of the VI.

## Hardware Triggered

Figure 14-2 shows the diagram of the Simul AI/AO Buffered Trigger (E-series MIO) VI located in `labview\examples\daq\anlog_io\anlog_io.llb`.

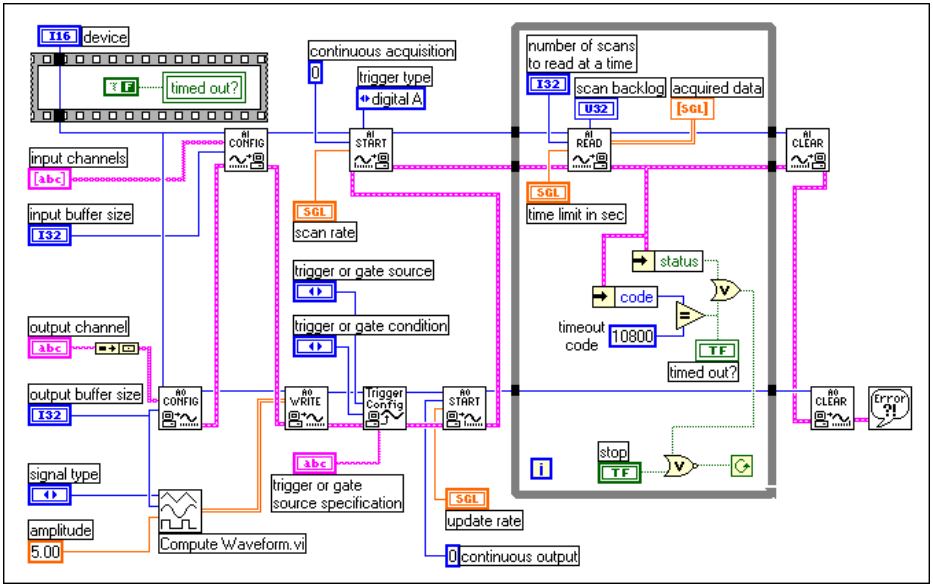



**Figure 14-2.** Simultaneous Input/Output Using the Simul AI/AO Buffered Trigger (E-series MIO) VI

Although this VI is similar to the example in Figure 14-1, it is more advanced because it uses a hardware trigger. The waveform acquisition trigger is set up with the **trigger type** input to the AI Start VI set to `digital A` (start), and by default this trigger is expected on the PFI0 pin. Hardware triggering for waveform generation requires an additional VI. The AO Trigger and Gate Config VI is an advanced analog output VI for E Series boards only. The trigger parameters are set using three inputs. The **trigger or gate source** is used to choose the source of your trigger, such as a PFI pin or a RTSI pin. The **trigger or gate source specification** is used in conjunction with the **trigger or gate source** to choose which PFI or RTSI pin number to use, such as 0 through 9 for a PFI pin. The **trigger or gate condition** is used to select a rising or falling trigger edge. The default analog output trigger for this example is a rising edge on PFI0. Because this is the same pin as the analog input trigger, the waveform

acquisition and generation starts simultaneously. However, they are not controlled by independent counter/timers, so you can run them at different rates.

For a complete description, instructions, and I/O connections for this VI, select **Windows»Show VI Info** from the front panel of the VI.

# Using Legacy MIO Boards

Legacy MIO devices, such as the AT-MIO-16, have a total of five counters, of which two or more can be used for data acquisition and generation. However, certain counters are dedicated to certain tasks, and you must be aware of this as you design your system.

## Software Triggered

Figure 14-3 shows the diagram of the Simul AI/AO Buffered (legacy MIO) VI located in `labview\examples\daq\anlog_io\anlog_io.llb`.

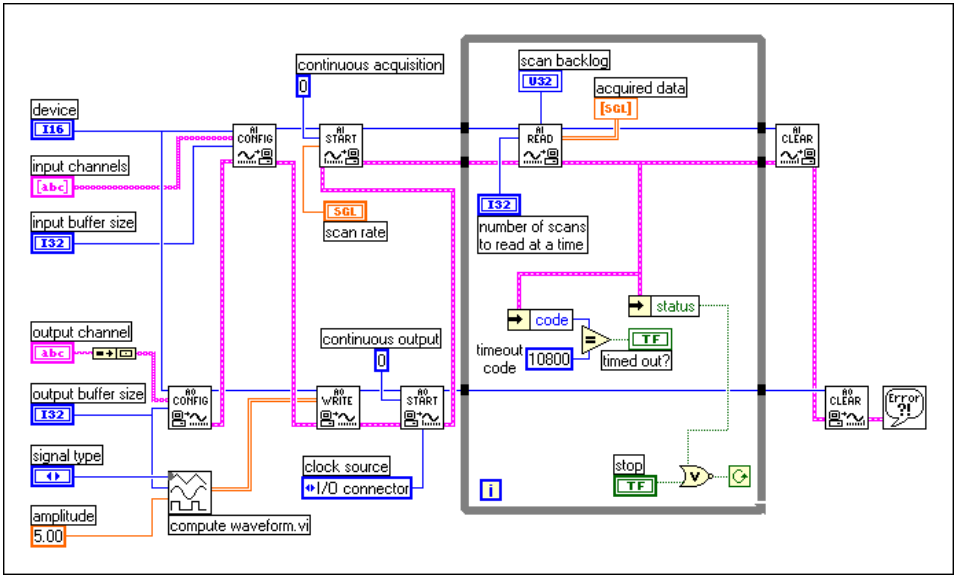



**Figure 14-3.** Simultaneous Input/Output Using the Simul AI/AO Buffered (Legacy MIO) VI

Because legacy MIO-type boards have only one clock available for signal acquisition (scan timing) and generation (update timing), the same clock is used for both. The acquisition uses `counter 2` by default. The generation is set up to use the I/O connector at the **clock source** input to the AO Start VI. Because the I/O connector **scan clock** input is the OUT2 pin, which already has the acquisition timing signal on it, no external clock wiring is required. The result is that the waveform acquisition and generation start simultaneously and occur at the same rate using the same clock. Your waveform generation occurs at the same rate as the scan rate you choose for waveform acquisition.

For a complete description, instructions, and I/O connections for this VI, select **Windows»Show VI Info** from the front panel of the VI.

## Hardware Triggered

Figure 14-4 shows the diagram of the Simul AI/AO Buffered Trigger (legacy MIO) VI located in `labview\examples\daq\anlog_io\anlog_io.llb`.

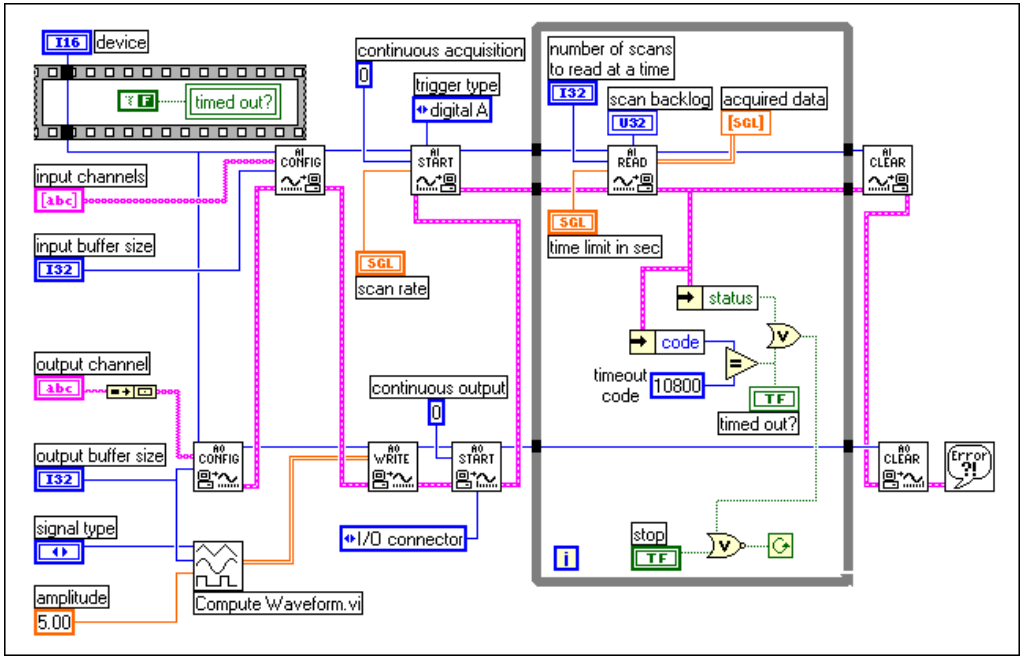



**Figure 14-4.** Simultaneous Input/Output Using the Simul AI/AO Buffered Trigger (Legacy MIO) VI

The only difference between this example VI and the example in Figure 14-3 is the **trigger type** input to the AI Start VI is set to `digital A` (start) trigger. This sets up the waveform acquisition for a digital trigger. Because the waveform generation uses the same counter/timer as the waveform acquisition, it also is dependent on the digital trigger.

For a complete description, instructions, and I/O connections for this VI, select **Windows»Show VI Info** from the front panel of the VI.

# Using Lab/1200 Boards

Lab/1200 boards, such as the Lab-PC-1200 or the DAQCard-1200, also can perform simultaneous waveform acquisition and generation. The approach is similar to the previous descriptions. Refer to the examples Simul AI/AO Buffered (Lab/1200) VI and Simul AI/AO Buffered Trigger (Lab/1200) VI located in `labview\examples\daq\anlog_io\anlog_io.llb` to see how this acquisition and generation is performed.

# Part IV

# Getting Square with Digital I/O

This section describes basic concepts about how to use digital signals with data acquisition in LabVIEW, including immediate and handshaked digital I/O.

Part IV, *Getting Square with Digital I/O*, contains the following chapters:

- Chapter 15, *Things You Should Know about Digital I/O*, explains basic concepts of digital I/O.
- Chapter 16, *When You Need It Now— Immediate Digital I/O*, explains how to use digital lines to acquire and generate data immediately.
- Chapter 17, *Shaking Hands with a Digital Partner*, shows you how you can synchronize digital data transfers between your DAQ devices and instruments.

# 15

# Things You Should Know about Digital I/O

Digital I/O interfaces are often used to control processes, generate patterns for testing, and communicate with peripheral equipment like heaters, motors, and lights. Digital I/O components on DAQ devices and SCXI modules consist of hardware parts that generate or accept binary on/off signals. As shown in the diagram below, all digital lines are grouped into ports on DAQ devices and banks on SCXI modules. The number of digital lines per port or bank is specific to the particular device or module used, but most ports or banks consist of four or eight lines. Except for the 6533 in immediate mode, TIO-10, and E Series devices, all lines within the same port or bank must all be of the same direction (either input or output), as shown in Figure 15-1. By writing to or reading from a port, you can set or retrieve simultaneously the states of multiple digital lines. Refer to Appendix B, *Hardware Capabilities*, of the *LabVIEW Function and VI Reference Manual*, your hardware user manual, or refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**, for port information on your device.

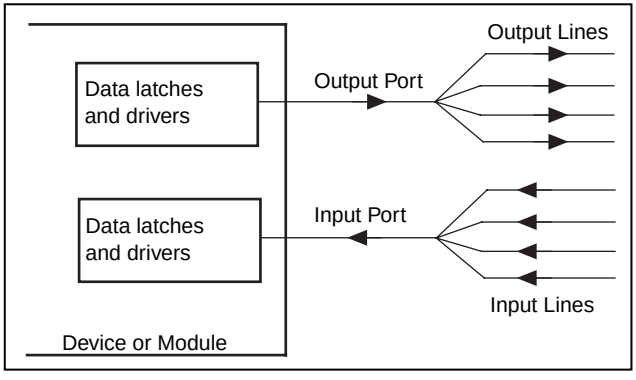



**Figure 15-1.** Digital Ports and Lines

# Types of Digital Acquisition/Generation

There are two types of digital acquisition/generation—*nonlatched* (or *immediate*) and *latched* (or *handshaked*). With nonlatched or immediate digital I/O, your system updates the digital lines immediately. Latched or handshaked digital I/O is when a device or module accepts or transfers data after a digital pulse has been received. There are two types of latched (handshaked) digital I/O: nonbuffered and buffered. Not all devices and modules support latched (handshaked) digital I/O. Refer to the hardware tables in Appendix B, *Hardware Capabilities*, of the *LabVIEW Function and VI Reference Manual*, your hardware manual, or refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**, to see if your device or module supports it.

For specific information about the Digital I/O VIs, refer to Chapter 14, *Introduction to the LabVIEW Data Acquisition VIs*, in the *LabVIEW Function and VI Reference Manual*, or refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**.

# 16

# When You Need It Now—Immediate Digital I/O

This chapter focuses on transferring data across a single port. The most common way to use digital lines is with nonlatched (immediate) digital I/O. All DAQ devices and SCXI modules with digital components support this mode.

When your program calls a function in nonlatched digital I/O mode, LabVIEW immediately updates the digital line or port output state or returns the current digital value of an input line, depending on the digital line direction. LabVIEW inputs or outputs only one value on each digital line in this mode. You can completely configure port (and for some devices, line) direction in software, and you can switch directions repeatedly in a program.

An example of when you might use nonlatched (immediate) digital I/O is in controlling or monitoring relays. You can also use multiple ports or groups of ports to perform digital I/O functions. In order to group digital ports, you must use Intermediate or Advanced VIs in LabVIEW. You can read more about grouping multiple digital ports in the next chapter, Chapter 17, *Shaking Hands with a Digital Partner*.

You can use the Easy Digital VIs for nonlatched digital I/O. Figure 16-1 shows the Easy VIs and their various inputs and outputs. The four Easy VIs can read data from or write data to a single digital line or to an entire port immediately. For an example of how to use the Easy Digital VIs, refer to the Read 1 Point from Digital Line and Write 1 Point to Digital Line VIs in `labview\examples\daq\digital\digio.llb`. Use the Easy Digital VIs for most digital testing purposes. All of the Easy Digital VIs have error reporting.

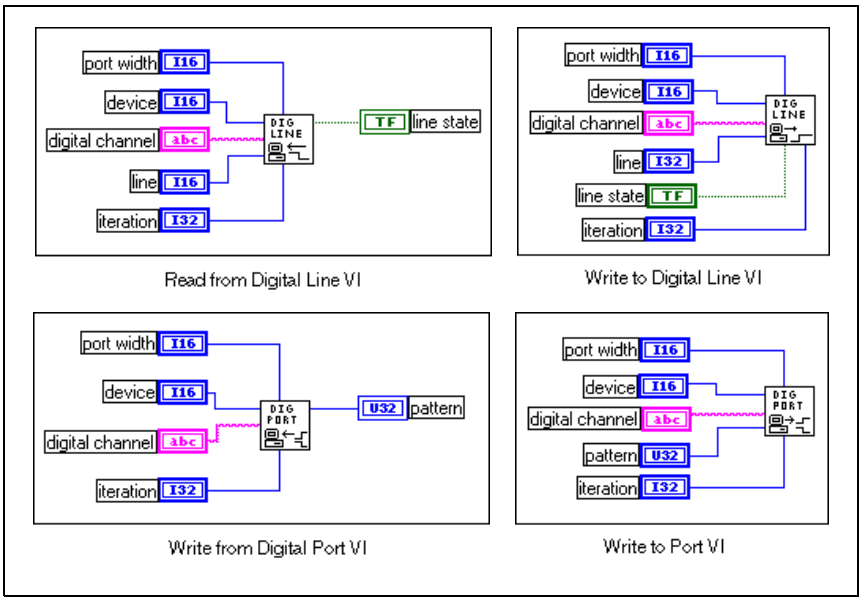



**Figure 16-1.** The Easy Digital VIs

If you have configured channels using the DAQ Channel Wizard, **digital channel** can consist of a digital channel name. The channel name may refer to either a port or a line in a port. You do not need to specify **device**, **line**, or **port width** because these inputs are not used by LabVIEW if a channel name is specified in **digital channel**. For more information about using the DAQ Channel Wizard to configure your channels, refer to the *Configuring Your Channels in NI-DAQ 5.x, 6.x* section of Chapter 2, *Installing and Configuring Your Data Acquisition Hardware*. For more information about using channel names, refer to the *Channel Name Addressing* section of Chapter 3, *Basic LabVIEW Data Acquisition Concepts*.

As an alternative, **digital channel** can consist of a port number. The port number specifies the port of digital lines that you will use during your digital operation. In this case, you must also specify **device**, **line**, and **port width** where applicable to further define your digital operation. The **device** input identifies the DAQ device you are using. The **line** input is an individual port bit or line in the port specified by **digital channel**. The **port width** input specifies the number of lines that are in the port you are using.

The **pattern** or **line state** is the value(s) you want to read from or write to a device. Pattern values can be displayed in decimal (default), hexadecimal, octal, or binary form. Refer to Chapter 9, *Numeric Controls and Indicators*, in the *G Programming Reference Manual* for instructions on how to change the display of a numeric control or indicator. The **iteration** input optimizes your digital operation. When **iteration** is `0` (the default value), LabVIEW calls the DIO Port Config VI (an Intermediate VI) to configure the port. If **iteration** is greater than `0`, LabVIEW uses the existing configuration, which improves performance. You can wire this input to an iteration terminal of a loop. Every time **iteration** is `0`, you call the DIO Port Config VI, which configures the port and, for many devices, resets the digital line values to their default values. If you want to use the same digital values from one loop iteration to another, set **iteration** to `0` only for the first iteration of the loop, then change it to a value greater than `0`.

If you are using an SCXI module for nonlatched digital I/O and are not using channel names, refer to the *SCXI Channel Addressing* section in Chapter 20, *Special Programming Considerations for SCXI*, for instructions on how to specify port numbers.

# 17

# Shaking Hands with a Digital Partner

If you want to pass a digital pattern after receiving a digital pulse, you should use *latched digital I/O*, also called *handshaking*. For example, you might want to acquire an image from a scanner. The scanner sends a pulse to your DAQ device after the image has been scanned and it is ready to transfer the data. Then, your DAQ device reads a digital pattern, perhaps 8, 16, or 32 bits in width. Your DAQ device then sends a pulse to the scanner to let it know the digital pattern has been read. The scanner sends out another pulse when it is ready to send another digital pattern. After your DAQ device receives this digital pulse, it reads the data. This process repeats until all the data is transferred. Handshaking gives you the ability to synchronize digital data transfer between your DAQ device and instrument.

Many DAQ devices support digital handshaking, including the following:

- DIO-32 devices
  - 6533 (DIO-32HS) devices
  - DIO-32F
- 8255-based devices
  - 6503 (DIO-24) devices
  - 6507/6508 (DIO-96) devices
  - 6025E devices
  - AT-MIO-16DE-10
  - 1200 Series devices
  - Lab Series devices

**Note** Combining channel names configured in the DAQ Channel Wizard and handshaking are not supported.

Another example of when you can use handshaking is testing the durability of a product prototype. Each durability test would be performed with a different piece of machinery for the same amount of time. For each test, you can turn the machinery on and off with a specific variation of handshaked digital I/O, known as *pattern I/O*. Internal counters would serve to generate the handshaking signal that initiates a digital transfer. Counters output digital pulses at a steady frequency. Thus, you can generate and retrieve patterns at a constant rate because the handshaking signal would be produced at a constant rate. However, you can use this rate only if the instrument or external hardware does not require a two-way REQ/ACK handshake for its data transfers. Only the DIO-32 devices (6533, DIO-32HS, and DIO-32F) support pattern I/O.

If you have an external signal controlling your digital I/O operation, you should connect the outside signal to the I/O connector or the RTSI connector. For more information on these connectors, refer to your hardware manual for your device. The names and functions of handshaking signals vary. For the DIO-32 devices, there are two main handshaking lines—the REQ (request) line and the ACK (acknowledge) line. REQ is an input indicating the external device is ready. ACK is an output indicating the DIO-32 device is ready. Burst mode on a DIO-32HS or 6533 device also uses a third handshaking signal, PCLK (peripheral clock).

For 8255-based DAQ devices that perform handshaking, there are four handshaking signals: Strobe Input (STB), Input Buffer Full (IBF), Output Buffer Full (OBF), and Acknowledge Input (ACK). You use the STB and IBF signals for digital input operations and the OBF and ACK signals for digital output operations. When the STB line is low, LabVIEW loads data into the DAQ device. After the data has been loaded, IBF is high, which tells the external device that the data has been read. For digital output, OBF is low while LabVIEW sends the data to an external device. After the external device receives the data, it sends a low pulse back on the ACK line. Check your DAQ device hardware manual for information on which digital ports can be configured for handshaking signals.

# Sending Out Multiple Digital Values

You can group multiple ports together so you can send more digital values out at a time. For DIO-32 devices, the ports in the group determine which handshaking lines are used. If the group includes port 0 or 1, handshaking occurs on the group 1 handshaking lines. Otherwise, if the group consists only of a combination of ports 2 and 3, handshaking occurs on the group 2 handshaking lines. In either case, the LabVIEW group number does not affect which handshaking lines are used.

For 8255 devices, the ports in the group and the order of the ports both affect which handshaking lines are used. If you want to group ports 0 and 1 and you list the ports in the order of `0:1`, then you should use the handshaking lines associated with port 1. In other words, always use the handshaking lines associated with the last port in the list. So, if the ports are listed `1:0`, then you should use the handshaking lines associated with port 0.

For 8255-based devices that perform handshaking, such as the DIO-24 or DIO-96, you must connect all the STB lines together if you are grouping ports for digital input, as shown in Figure 17-1. Connect only the IBF line of the last port in the port list to the other device. No connection is needed for the IBF signals for the other ports in the port list.

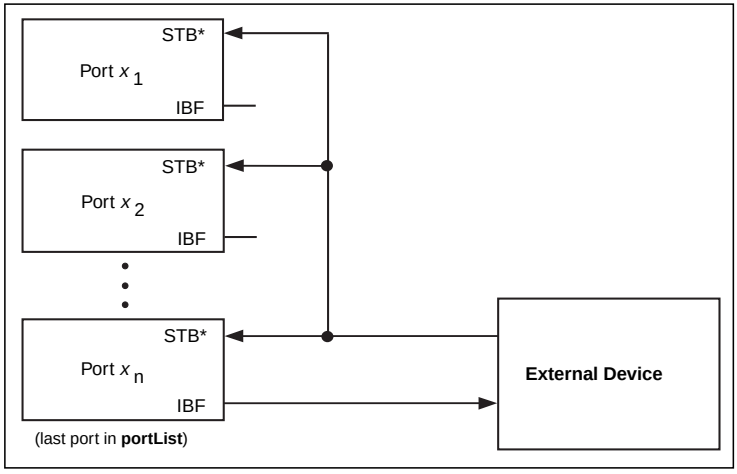



**Figure 17-1.** Connecting Signal Lines for Digital Input

If you group ports for digital output on an 8255-based device, connect only the handshaking signals of the last port in the port list, as shown in Figure 17-2.

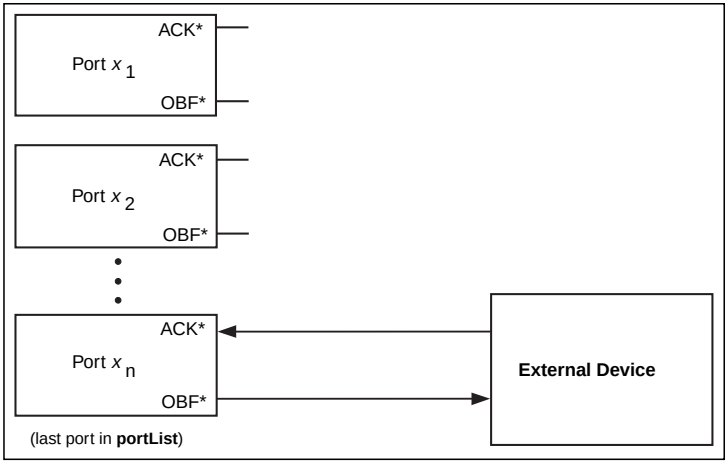



**Figure 17-2.** Connecting Digital Signal Lines for Digital Output

There are two types of digital handshaking: nonbuffered and buffered. Nonbuffered handshaking is similar to nonlatched digital I/O because LabVIEW updates the digital lines immediately after every digital or handshaked pulse.

**Note** For the 6533 (DIO-32HS) devices, LabVIEW returns immediately after storing data in its FIFO buffer.

With buffered handshaking, LabVIEW stores digital values in memory to be transferred after every handshaked pulse. Both nonbuffered and buffered handshaking transfer one digital value after each handshaked pulse. For basic digital applications, use nonbuffered handshaking. Use buffered handshaking when your application requires multiple handshaking pulses or high speeds. By using a buffer with multiple handshaking pulses, the software spends less time reading or writing data, leaving more time for other operations.

**Note** On an AT-DIO-32F device with nonbuffered handshaking, you can group 1, 2, or 4 ports together. For buffered handshaking on the AT-DIO-32F, you can group only 2 or 4 ports together.

You can use only Intermediate or Advanced Digital VIs for digital handshaking in LabVIEW. The Intermediate VIs work for most nonbuffered and buffered digital handshaking applications. However, for some DAQ devices, you might need a combination of Intermediate and Advanced VIs.

# Nonbuffered Handshaking

Nonbuffered handshaking occurs when your program transfers one digital value after receiving a digital pulse on the handshaking lines. LabVIEW does not store these digital values in computer memory. You should only use nonbuffered handshaking when you expect only a few digital handshaking pulses. For multiple-pulsed or high-speed applications, you should use buffered handshaking, which you can learn about in the next section of this chapter, *Buffered Handshaking*. Figure 17-3 shows an example of nonbuffered handshaking using the Intermediate VI, DIO Single Read/Write. In this example, LabVIEW reads the data from the digital port(s).

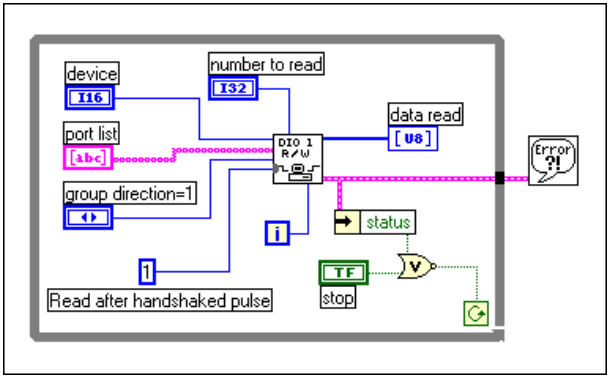



**Figure 17-3.** Nonbuffered Handshaking Using the DIO Single Read/Write VI

Typically, you want to put the DIO Single Read/Write VI inside a loop. You can use the **iteration** input (the terminal where the loop iteration is connected) to optimize your digital operation. When **iteration** is `0` (default), LabVIEW calls the Advanced VI, DIO Group Config, to configure the port(s). If **iteration** is greater than `0`, LabVIEW uses the existing configuration, which improves performance. Every time your program calls the DIO Group Config VI, the digital line values are reset to their default values. To set the digital line values once and keep the same values from one loop iteration to the next, set **iteration** to `0` on the first iteration of the loop, then set **iteration** to `1`. When **group direction** is equal

to 1 (default), all the ports listed in **port list** are treated as inputs. The number of elements in the **data read** input will be the same as the product of the number of ports in the group and the **number to read** input.

Figure 17-4 shows how you can use nonbuffered handshaking to write data. The programming flow resembles the read operation above. The **updates to write** array must contain as many elements as the number of ports multiplied by the number of values to write.

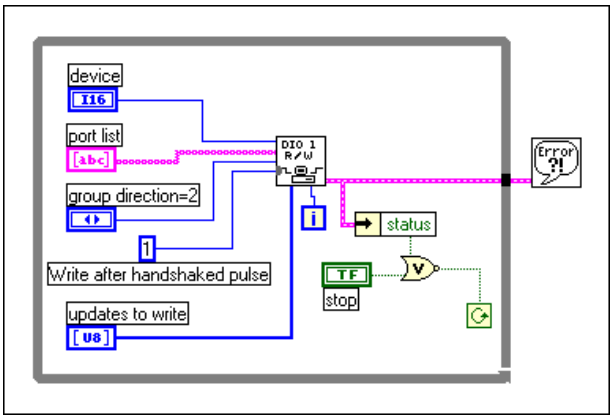



**Figure 17-4.** Nonbuffered Handshaking Using the DIO Single Read/Write VI

# Buffered Handshaking

Buffered handshaking allows you to store multiple points in computer memory. Use this technique if multiple pulses are expected on the handshaking lines. Buffered handshaking comes in two forms: simple and circular. You can use simple-buffered handshaking on all DAQ devices that support handshaking. You can perform circular-buffered handshaking only on the AT-DIO-32F and 6533 (DIO-32HS) devices. You can think of a simple buffer as a storage place in computer memory, where **buffer size** equals the number of updates multiplied by the number of ports.

A circular buffer differs from a simple buffer only in the way your program places the data into it and retrieves data from it. A circular buffer fills with data the same as a simple buffer, but when it gets to the end of the buffer LabVIEW returns to the beginning of the buffer and fills up the same buffer again. Use simple-buffered handshaking when you have a predetermined number of values to acquire or generate. Use circular-buffered handshaking when you want to acquire or generate data continuously.

## Simple Buffered Examples

The block diagram in Figure 17-5 uses the Intermediate VIs to perform pattern output using the DIO-32 devices. Example VIs included with LabVIEW are similar to the following block diagram. Notice the **port list** contains more than one port number, which means the ports are grouped together.

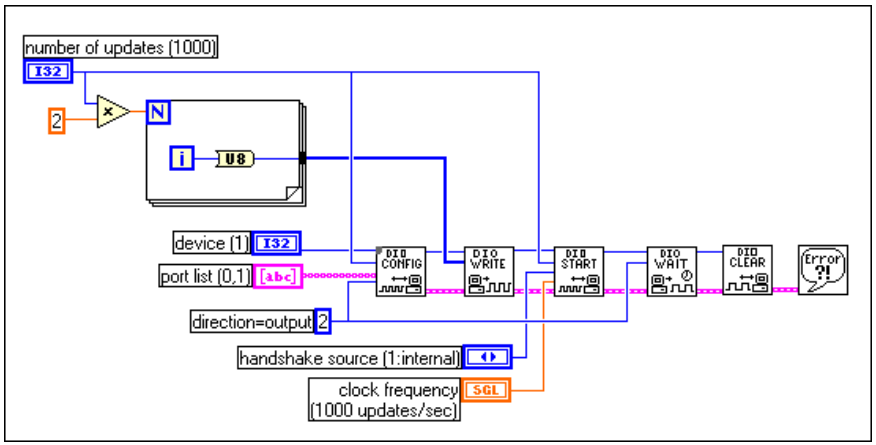



**Figure 17-5.** Buffered Output Using the DIO-32 Devices

The For Loop generates the digital data to output. The amount of data generated equals the number of ports in the **port list** multiplied by the **number of updates**.

The **direction** input specifies whether the ports are configured for input or output. The DIO Wait VI waits until the digital buffered input or output operation completes before returning to the main VI. The DIO Clear VI halts any transfers and clears the group port configuration. If you want an external source to supply the handshaking signals, you can specify the **handshake source** to be an external signal entering through the I/O connector (**handshake source** = 2). You only need to use the **clock frequency** if you are performing pattern I/O with an internal handshake source.

**Note** On 6533 (DIO-32HS) devices, use **handshake source = 2** for an external REQ source with ACK pulses (REQ/ACK handshaking). Use **handshake source = 4** for external REQ with no ACK pulses (pattern I/O with an external REQ source).

For DAQ devices other than the DIO-32 devices, you can use a VI similar to the one above to output digital data. The main difference is that **handshake source** must be external (2). Alternatively, you can use an Advanced Digital VI, DIO Buffer Control, instead of the DIO Start VI, as shown in Figure 17-6. The DIO Start VI contains Digital Clock Config and Digital Mode Config VIs that are useful with DIO-32 devices. You do not need to use the **handshake source** and **clock frequency** inputs, because the handshaking signal source is always external.

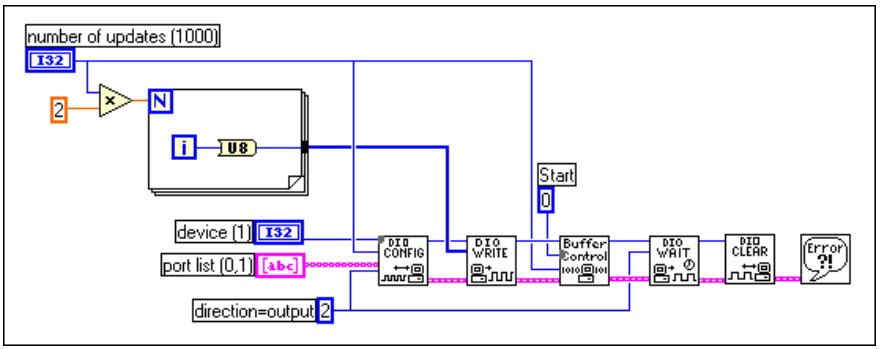



**Figure 17-6.** Buffered Output Using DAQ Devices (Other Than DIO-32 Series Devices)

Reading information is similar to writing data when using digital handshaking. In the example shown in Figure 17-7, the VI is reading data into the DIO-32 devices while using external handshaking. For the DIO-32 devices, the DIO Config VI can set or change the handshaking mode, for instance, active-high or active-low request signals.

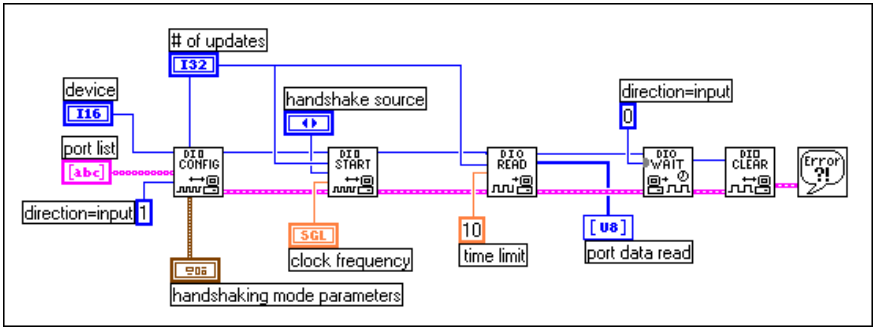



**Figure 17-7.** Buffered Input Using DIO-32 Devices

For the other devices that support digital handshaking, the example would be the same as above except the **handshaking mode** input would be deleted from the DIO Config VI and the DIO Start VI could be replaced with the DIO Buffer Control VI. Also, you do not need the **handshake source** and **clock frequency** inputs for most devices, because the handshaking signal source is always external. Figure 17-8 shows the VI used for all DAQ devices other than the DIO-32 devices.

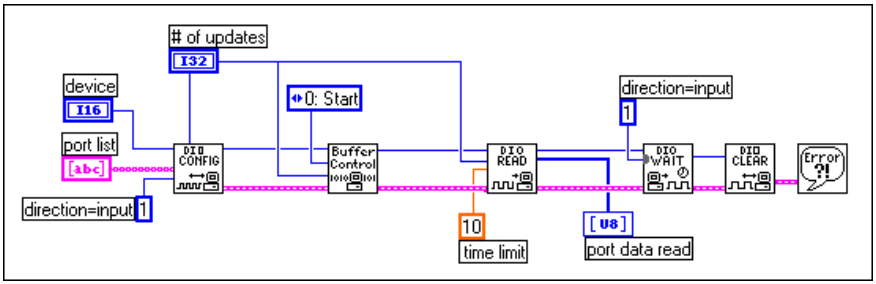



**Figure 17-8.** Buffered Input Using DAQ Devices (Other than DIO-32 Devices)

## Circular-Buffered Examples

Circular-buffered handshaking is similar to simple-buffered handshaking in that both types of handshaking place data in a buffer. However, a circular-buffer application returns to the beginning of the buffer when it reaches the end and fills the same buffer again.

**Note** Circular-buffered handshaking works only on the AT-DIO-32F and 6533 (DIO-32HS) devices.

Figure 17-9 shows an example of a circular-buffered application. In this example, you are reading or writing digital values continually until you stop the VI or an error occurs. You can have an internal or external **handshake source**. If your **handshake source** is internal, remember to specify the rate at which you read values with the **clock frequency**. **scan backlog** specifies how many values are left in the buffer after you read. The **number read** input indicates the total number of values that have been read from the buffer because the VI started executing.

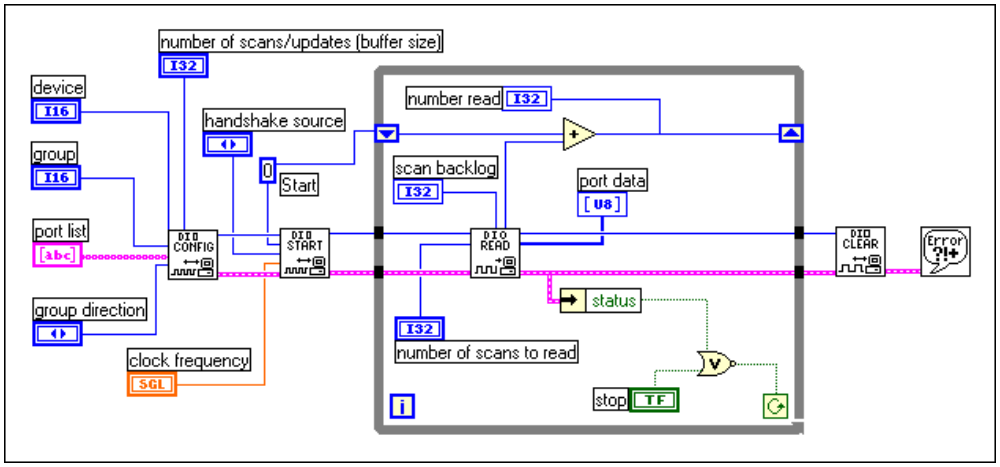



**Figure 17-9.** Digital Handshaking Using a Circular Buffer

Digital handshaking, both buffered and nonbuffered, inputs or outputs digital patterns only after your computer receives a digital pulse. Not all DAQ devices support digital handshaking. The DIO-32 devices have internal as well as external handshaking signals and support circular-buffered I/O. Other DAQ devices that support handshaking accept only external handshaking signals. Use digital handshaking when you need to generate or retrieve a digital pattern after a digital event, or pulse, is detected.

# Part V

# SCXI—Getting Your Signals in Great Condition

This section contains basic information about setting up and using SCXI modules with your data acquisition application, special programming considerations, common SCXI applications, and calibration information.

Part V, *SCXI—Getting Your Signals in Great Condition*, contains the following chapters:

- Chapter 18, *Things You Should Know about SCXI*, includes basic concepts on how to use SCXI modules with LabVIEW for data acquisition.
- Chapter 19, *Hardware and Software Setup for Your SCXI System*, explains how to set up your SCXI hardware to work with data acquisition in LabVIEW.
- Chapter 20, *Special Programming Considerations for SCXI*, describes special programming considerations for SCXI in LabVIEW, including channel addressing, gains (limit settings), and settling time.
- Chapter 21, *Common SCXI Applications*, covers example VIs for analog input, analog output, and digital SCXI modules.
- Chapter 22, *SCXI Calibration—Increasing Signal Measurement Precision*, teaches you how to calibrate SCXI modules and shows you where LabVIEW stores your calibration constants.

# 18

# Things You Should Know about SCXI

SCXI is a highly expandable signal conditioning system. The next few chapters describe the basic concepts of signal conditioning, the setup procedure for SCXI hardware, the hardware operating modes, the procedure for software installation and configuration, the special programming considerations for SCXI in LabVIEW, and some common SCXI applications.

**Note** For a better understanding of signal conditioning concepts, the chapters in Part V, *SCXI—Getting Your Signals in Great Condition*, refer to SCXI. However, the concepts and techniques discussed in these chapters also apply to VME eXtension for Instrumentation Signal Conditioning (VXI-SC).

## What Is Signal Conditioning?

Transducers can generate electrical signals to measure physical phenomena, such as temperature, force, sound, or light. Table 18-1 lists some common transducers.

**Table 18-1.** Phenomena and Transducers

| Phenomena | Transducer |
|---|---|
| Temperature | Thermocouples<br>Resistance temperature detectors (RTDs)<br>Thermistors<br>Integrated circuit sensor |
| Light | Vacuum tube photosensors<br>Photoconductive cells |
| Sound | Microphone |
| Force and pressure | Strain gauges<br>Piezoelectric transducers<br>Load cells |

**Table 18-1.** Phenomena and Transducers (Continued)

| Phenomena | Transducer |
|---|---|
| Position (displacement) | Potentiometers<br>Linear voltage differential transformer (LVDT)<br>Optical encoder |
| Fluid flow | Head meters<br>Rotational flowmeters<br>Ultrasonic flowmeters |
| pH | pH electrodes |

To measure signals from transducers, you must convert them into a form that a DAQ device can accept. For example, the output voltage of most thermocouples is very small and susceptible to noise. Therefore, you may need to amplify and/or filter the thermocouple output before digitizing it. The manipulation of signals to prepare them for digitizing is called *signal conditioning*. The following are some common types of signal conditioning:

- Amplification
- Isolation
- Filtering
- Transducer excitation
- Linearization

Figure 18-1 shows some common types of transducers/signals and the required signal conditioning for each.

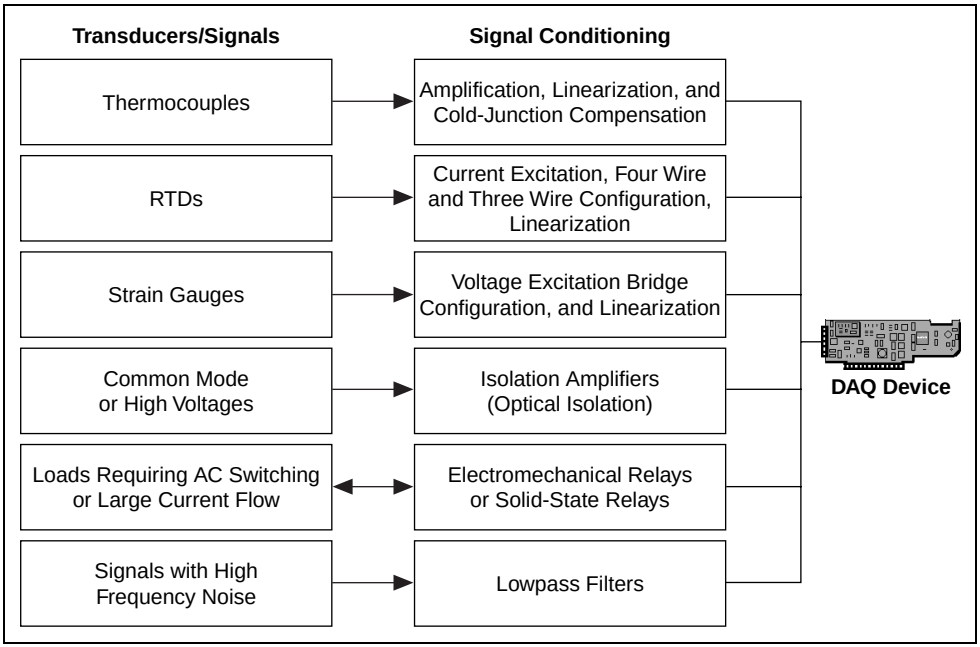



**Figure 18-1.** Common Types of Transducers/Signals and Signal Conditioning

# Amplification

The most common type of signal conditioning is *amplification*. Amplifying electrical signals improves the accuracy of the resulting digitized signal and reduces noise.

For the highest possible accuracy, amplify the signal so the maximum voltage swing equals the maximum input range of the ADC, or digitizer. Your system should amplify low-level signals at the DAQ device or at the SCXI module located nearest to the signal source, as shown in Figure 18-2.

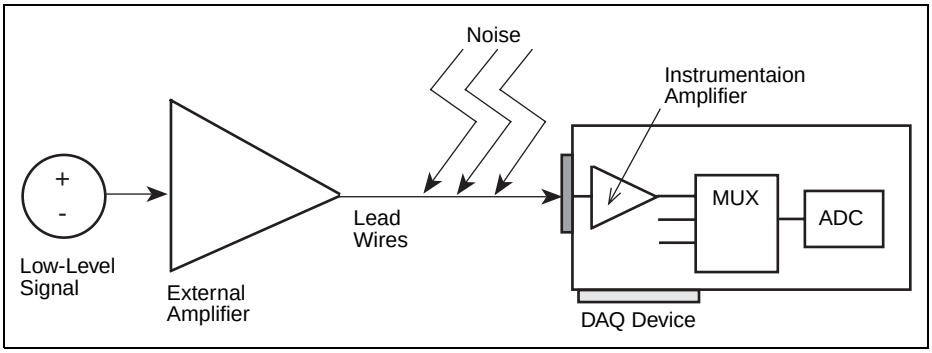



**Figure 18-2.** Amplifying Signals near the Source to Increase Signal-to-Noise Ratio (SNR)

**Note** You can minimize noise that lead wires pick up by using shielded cables or a twisted pair of cables, and by minimizing wire length. Also, keeping signal wires away from AC power cables and monitors will help reduce 50 Hz or 60 Hz noise.

If you amplify the signal at the DAQ device, the signal is measured and digitized with noise that may have entered the lead wires. However if you amplify the signal close to the signal source with an SCXI module, noise has a less destructive effect on the signal. In other words, the digitized representation is a better reflection of the original low-level signal. For more information, consult *Application Note 025, Field Wiring and Noise Considerations for Analog Signals.* You can access this note from the National Instruments web site, `www.natinst.com`.

# Isolation

Another common way to use SCXI is to isolate the transducer signals from the computer for safety purposes. When the signal being monitored contains large voltage spikes that could damage the computer or harm the operator, you should not directly connect the signal to a DAQ device without some type of isolation.

Another reason for isolation is to make sure the measurements from the DAQ device are not affected by differences in ground potentials. When the DAQ device and the signal are not referenced to the same ground potential, a ground loop may occur. Ground loops can cause an inaccurate representation of the measured signal. If the potential difference between the signal ground and the DAQ device ground is large, then damage may even occur to the measuring system. Using isolated SCXI modules eliminates the ground loop and ensures that the signals are accurately measured.

# Filtering

Signal conditioning systems can filter unwanted signals or noise from the signal you are trying to measure. You can use a noise filter on low-rate (or slowly changing) signals, like temperature, to eliminate higher-frequency signals that can reduce the accuracy of the digitized signal. A common use of a filter is to eliminate the noise from a 50 or 60 Hz AC power line. A lowpass filter of 4 Hz, which exists on several SCXI modules, is suitable for removing the 50 or 60 Hz AC noise from signals sampled at low rates. A lowpass filter eliminates all signal frequency components above the cutoff frequency. The SCXI-1141, SCXI-1142, and SCXI-1143 modules have lowpass filters that have software-selectable cutoff frequencies from 10 Hz to 25 kHz.

# Transducer Excitation

Signal conditioning systems can generate excitation for some transducers. Strain gauges and RTDs require external voltage and currents, respectively, to excite their circuitry into measuring physical phenomena. This type of excitation is similar to a radio that needs power to receive and decode audio signals. Some plug-in DAQ devices and SCXI modules, including the SCXI-1121 and SCXI-1122 modules, provide the necessary excitation for transducers.

# Linearization

Many transducers, such as thermocouples, have a nonlinear response to changes in the physical phenomena being measured. LabVIEW can linearize the voltage levels from transducers, so the voltages can be scaled to the measured phenomena. LabVIEW provides simple scaling functions to convert voltages from strain gauges, RTDs, thermocouples, and thermistors.

For specific information about the VIs you can use with your SCXI module in LabVIEW, refer to Chapter 29, *Calibration and Configuration VIs*, in the *LabVIEW Function and VI Reference Manual*, or refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**.

# 19

# Hardware and Software Setup for Your SCXI System

SCXI hardware conditions signals close to the signal source and increases the number of analog and digital signals that a DAQ device can analyze. With-PC compatible computers, SCXI can be configured in two ways—a front-end signal conditioning system for plug-in DAQ devices, or an external data acquisition and control system. Furthermore, when SCXI is configured as an external data acquisition and control system, it can be connected to the computer's parallel port using an SCXI-1200, or the computer's serial port using either an SCXI-2000 remote chassis or an SCXI-2400 remote communications module in an SCXI-100*x* chassis. For Macintosh computers, SCXI hardware can only be used as a front-end signal conditioning system for plug-in DAQ devices. Figure 19-1 demonstrates these configurations.

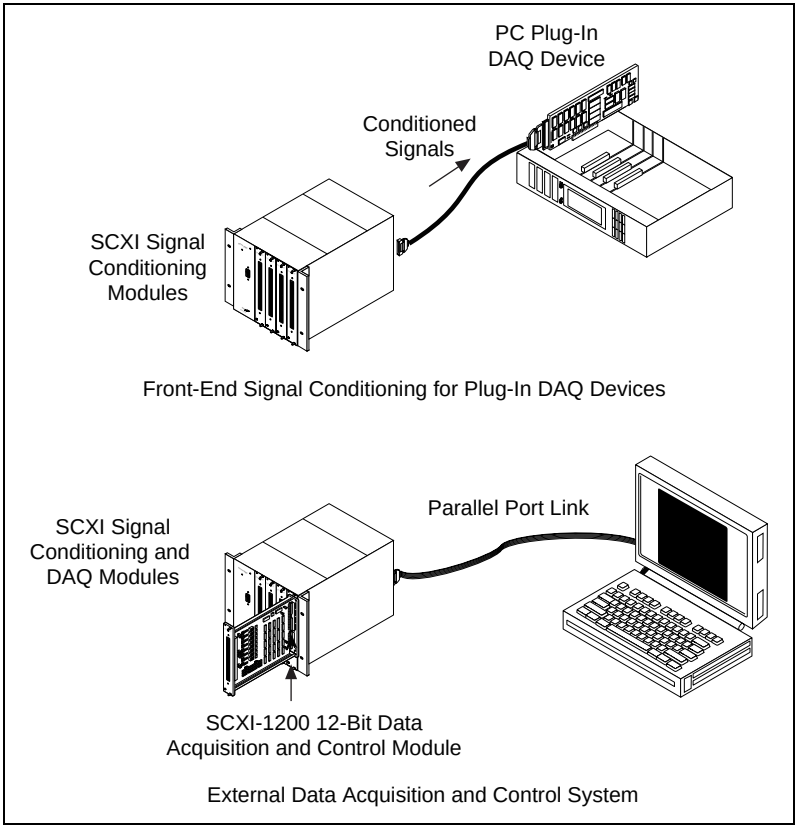



**Figure 19-1.** SCXI System

Figure 19-2 shows the components of an SCXI system. An SCXI system consists of an SCXI chassis that houses signal conditioning modules, terminal blocks that plug directly into the front of the modules, and a cable assembly that connects the SCXI system to a plug-in DAQ device or the parallel or serial port of a computer. If you are using SCXI as an external DAQ system where there are no plug-in DAQ devices, you can use the SCXI-1200 module, which is a multifunction analog, digital, and timing I/O (counters) module. The SCXI-1200 can control several SCXI signal conditioning modules installed in the same chassis. The functionality of the SCXI-1200 module is similar to the plug-in 1200 series devices.

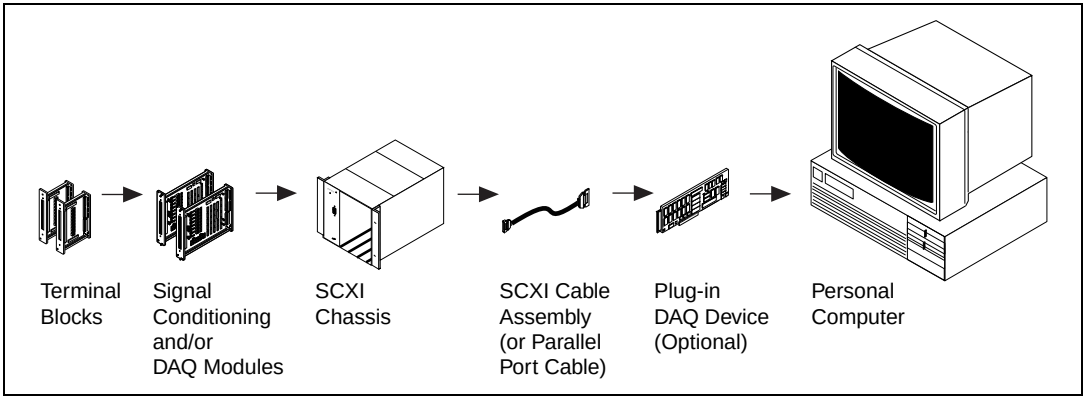



**Figure 19-2.** Components of an SCXI System

Refer to the SCXI tables in Appendix B, *Hardware Capabilities*, of the *LabVIEW Function and VI Reference Manual*, for tables containing specifications for all the SCXI modules, or refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**. This appendix also includes a list of all the SCXI modules and the compatible terminal blocks.

**Note** For information on how to set up each module and transducer, consult your hardware user manuals and the *Getting Started with SCXI* manual.

How do you transfer data from the SCXI chassis to the DAQ device or parallel or serial port? Figure 19-3 shows a diagram of an SCXI chassis. When you use SCXI as a front-end signal conditioning system, the analog and digital bus backplane, also known as the SCXIbus, transfers analog and/or digital data to the DAQ device. Some of the analog and digital lines on the DAQ device are reserved for SCXI chassis communication. To find out which lines are reserved on your device, refer to the tables in Appendix B, *Hardware Capabilities*, in the *LabVIEW Function and VI Reference Manual*, or refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**.

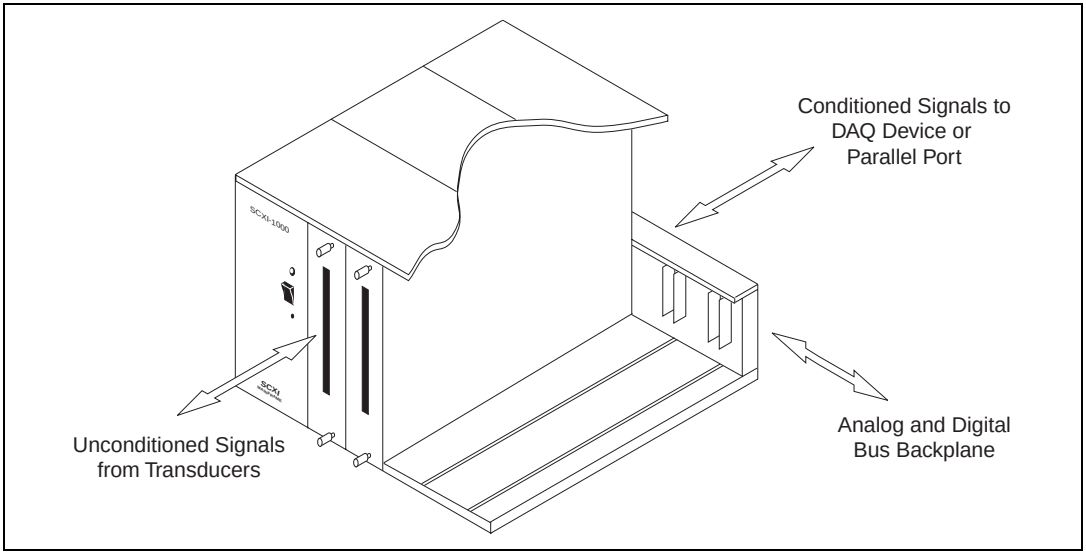



**Figure 19-3.** SCXI Chassis

When you use SCXI as an external DAQ system, only some of the digital I/O lines of the DAQ device are reserved for SCXI chassis communication when other modules are present. The DAQ device digitizes any analog input data and transfers it back to the computer through the parallel or serial port.

**Note** When using remote SCXI, be aware of the sampling rate limitations from the data being sent over the serial port. To reduce delays in serial port communication, National Instruments recommends that you use the fastest baud rate possible for your computer's serial port. If you have a 16550 or compatible universal asynchronous receiver-transceiver (UART), you can use baud rates up to 57,600 baud. If you have an 8250 or compatible UART, you can only use up to 19,200 baud.

# SCXI Operating Modes

The SCXI operating mode determines the way that DAQ devices access signals. There are two basic operating modes for SCXI modules, multiplexed and parallel. You designate the mode in the **operating mode** input in the configuration utility or Measurement & Automation Explorer. Also, you may have to set up jumpers on the module for the correct operating mode. Check your SCXI module user manual for more information.

**Note** National Instruments recommends that you use the multiplexed mode for most purposes.

## Multiplexed Mode for Analog Input Modules

When an analog input module operates in multiplexed mode, all of its input channels are multiplexed to one module output. When you cable a DAQ device to a multiplexed analog input module, the DAQ device has access to that module's multiplexed output, as well as all other modules in the chassis through the SCXIbus. The analog input VIs route the multiplexed analog signals on the SCXIbus for you transparently. So, if you operate all modules in the chassis in multiplexed mode, you only need to cable one of the modules directly to the DAQ device.

**Note** MIO/AI devices, and Lab-PC+ and 1200 devices support multiple-channel and multiple-scan acquisitions in multiplexed mode. The Lab-NB and Lab-LC, LPM devices, and DAQCard-700 support only single-channel or single-scan acquisitions in multiplexed mode.

When you connect a DAQ device to a multiplexed module, the multiplexed output of the module (and all other multiplexed modules in the chassis) appears at analog input channel 0 of the DAQ device by default.

### Multiplexed Mode for the SCXI-1200 (Windows)

In multiplexed mode, the SCXI-1200 can access the analog signals on the SCXI bus. The DAQ VIs can multiplex the channels of analog input modules and send them on the SCXIbus. This means that if you configure the SCXI-1200 for multiplexed mode, you can read the multiplexed output from other SCXI analog input modules in the chassis.

**Note** The SCXI-1200 only reads analog input module channels configured in multiplexed mode, not in parallel mode.

Make sure that you change the jumper in the SCXI-1200 to the ground position to connect the SCXI-1200 and SCXIbus grounds together. Refer to the *SCXI-1200 User Manual* for more details.

## Multiplexed Mode for Analog Output Modules

Because LabVIEW communicates with the multiplexed modules over the SCXIbus backplane, you must cable only one multiplexed module in each chassis to a DAQ device to communicate with any multiplexed modules in the chassis.

## Multiplexed Mode for Digital and Relay Modules

Multiplexed mode is referred to as *serial mode* in the digital and relay module hardware manuals. When you operate your digital or relay module in multiplexed mode, LabVIEW communicates the module channel states serially over the SCXIbus backplane.

## Parallel Mode for Analog Input Modules

When an analog input module operates in parallel mode, the module sends each of its channels directly to a separate analog input channel of the DAQ device cabled to the module. You *cannot* multiplex parallel outputs of a module on the SCXIbus. You must cable a DAQ device directly to a module in parallel mode to access its input channels. In this configuration, the number of channels available on the DAQ device limits the total number of analog input channels. In some cases, however, you can cable more than one DAQ device to separate modules in an SCXI chassis. For example, you can use two NB-MIO-16X or AT-MIO-16E-2 devices operating in parallel mode and cable each one to a separate SCXI-1120 module in the chassis. For more information on how to configure the cabled device, refer to the help file for Measurement & Automation Explorer by selecting **Help»Help Topics»DAQ Help** or the *Installing and Configuring*

*Your SCXI Chassis* section in Chapter 2, *Installing and Configuring Your Data Acquisition Hardware*, for the Macintosh.

By default, when a module operates in parallel mode, the module sends its channel 0 output to differential analog input channel 0 of the DAQ device, the channel 1 output to analog input channel 1 of the DAQ device, and so on.

When you use the analog input VIs, specify the correct onboard channel for each parallel SCXI channel. If you are using a range of SCXI channels, LabVIEW assumes the onboard channel numbers match the SCXI channel numbers. Refer to the *SCXI Channel Addressing* section in Chapter 20, *Special Programming Considerations for SCXI*, for the proper SCXI channel syntax.

### Parallel Mode for the SCXI-1200 (Windows)

In parallel mode, the SCXI-1200 reads only its own analog input channels. The SCXI-1200 does not have access to the analog bus on the SCXI backplane in parallel mode. You should use parallel mode if you are not using other SCXI analog input modules in the chassis with the SCXI-1200.

## Parallel Mode for Digital Modules

When you operate a digital module in parallel mode, the digital lines on your DAQ device directly drive the individual digital channels on your SCXI module. You must cable a DAQ device directly to every module operated in parallel mode.

You may want to use parallel mode instead of multiplexed mode for faster updating or reading of the SCXI digital channels. For the fastest performance in parallel mode, you can use the appropriate onboard port numbers instead of the SCXI channel string syntax in the digital VIs. Refer to the hardware tables in Appendix B, *Hardware Capabilities*, in the *LabVIEW Function and VI Reference Manual* for the digital ports used in parallel mode on each DAQ device, or refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**.

**Note** If you are using a 6507/6508 (DIO-96), an AT-MIO-16D, or an AT-MIO-16DE-10 device, you can also operate a digital module in parallel mode using the digital ports on the second half of the NB5 or R1005050 ribbon cable (lines 51–100). Therefore, the DIO-96 can operate two digital modules in parallel mode, one module using the first half of the ribbon cable (lines 1–50), and another module using the second half of the ribbon cable (lines 51–100).

# SCXI Software Installation and Configuration

After you assemble your SCXI system, you must run Measurement & Automation Explorer or the configuration utility to enter your SCXI configuration. LabVIEW needs the configuration information to program your SCXI system correctly. Refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware*.

# 20

# Special Programming Considerations for SCXI

When you want LabVIEW to acquire data from SCXI analog input channels, you use the analog input VIs in the same way that you acquire data from onboard channels. You also read and write to your SCXI relays and digital channels using the digital VIs in the same way that you read and write to onboard digital channels. You can write voltages to your SCXI analog output channels using the analog output VIs. The following sections describe special programming considerations for SCXI in LabVIEW, including channel addressing, gains (limit settings), and settling time.

**Note** This chapter does not apply if you use the DAQ Channel Wizard to configure your channels. In NI-DAQ 6.5 or higher, the DAQ Channel Wizard is part of Measurement & Automation Explorer. If you use the DAQ Channel Wizard, you address SCXI channels the same way you address onboard channels—by specifying the channel name(s). LabVIEW configures your hardware by selecting the best input limits and gain for the named channel based on the channel configuration. For more information about using the DAQ Channel Wizard to configure your channels, see the *Configuring Your Channels in NI-DAQ 5.x, 6.x* section of Chapter 2, *Installing and Configuring Your Data Acquisition Hardware*. For more information about using channel names, refer to the *Channel Name Addressing* section of Chapter 3, *Basic LabVIEW Data Acquisition Concepts*.

## SCXI Channel Addressing

If you operate a module in parallel mode, you can specify an SCXI channel either by specifying the corresponding onboard channels or by using the SCXI channel syntax described in this section. If you operate the modules in multiplexed mode, you must use the SCXI channel syntax.

An SCXI channel number has four parts: the onboard channel (optional), the chassis ID, the module number, and the module channel.

In the following table of examples, x is any chassis ID, y is any module number, a is any module channel, and b is any module channel greater than a. z is the onboard channel from which the conditioned data is

retrieved. If you operate in multiplexed mode, analog input channel 0 reads the data from the first cabled chassis. If you use VXI-SC submodules, LabVIEW ignores the onboard channel, because VXI-DAQ provides a special channel for retrieving data from submodules.

| Channel List Element | Channel Specified |
|---|---|
| `obz!scx!mdy!a` | Channel `a` on the module in slot `y` of the chassis with ID `x` is multiplexed into onboard channel `z`. |
| `obz!scx!mdy!a:b` | Channels `a` through `b` inclusive on the module in slot `y` of the chassis with ID `x` are multiplexed into onboard channel `z`. |
| `obz!scx!mdy!(a,b,c)` | Channels `a`, `b`, and `c` (nonconsecutive) on the module in slot `y` of the chassis with ID `x` are multiplexed into onboard channel `z`. (Only on certain SCXI modules such as the SCXI-1125.) |

The **channel** input for DAQ VIs is either a string (with the Easy I/O VIs) or an array of strings. Each string value can list only the channels for one module. With the array structure for channel values, you can list the channels for several modules. Therefore, for one scanning operation, you can scan several modules. You can scan an arbitrary number of channels for each module, but the channels of each module must be scanned in consecutive, ascending order.

**Note** You do not need the SCXI channel string syntax to access channels on the SCXI-1200 module. Use `0` for channel 0, `1` for channel 1, and so on. The SCXI-1200 module is identified by its logical device number.

**Note** When you connect any type of SCXI module to a DAQ device, certain analog input and digital lines on the DAQ device are reserved for SCXI control. On MIO Series devices, lines 0, 1, and 2 are unavailable for general purpose digital I/O. On MIO E Series devices, lines 0, 1, 2, and 4 are unavailable for general purpose digital I/O. For more channel information, refer to the LabVIEW *Online Reference* by selecting **Help»Online Reference**.

For the fastest performance in parallel mode on digital modules, use the appropriate onboard port numbers instead of the SCXI channel string syntax in the digital VIs.

# SCXI Gains

SCXI modules provide higher analog input gains than those available on most DAQ plug-in devices.

**Note** Before reading this section, you should have already read the *Limit Settings* section in Chapter 3, *Basic LabVIEW Data Acquisition Concepts*.

Enter the gain jumper settings in Measurement & Automation Explorer or the NI-DAQ Configuration utility for each channel on each module with jumpered gains. LabVIEW stores these gain settings and uses them to scale the input data. When you use the **input limits** control of the analog input VIs, LabVIEW chooses onboard gains that complement the jumpered SCXI gains to achieve the given input limits as closely as possible.

For analog input modules with programmable gains, LabVIEW uses the gain setting you enter in Measurement & Automation Explorer or the NI-DAQ Configuration utility for each module as the *default gain* for that module. LabVIEW uses the default gain for the module whenever you leave the **input limits** terminal to the analog input VIs unwired, or if you enter `0` for your upper and lower input limits.

When you use the **input limits** to specify non-zero limits for a module with programmable gains, LabVIEW chooses the most appropriate SCXI gain for the given limits. LabVIEW selects the highest SCXI gain possible for the given limits, and then selects additional DAQ device gain if necessary.

If your module has programmable gains and only one gain for all channels and you are using an MIO/AI DAQ device, you can specify different input limits for channels on the same module by splitting up your channel range over multiple elements of the channel array, and using a different set of input limits for each element. LabVIEW selects one module gain suitable for all of the input limits for that module, then chooses different MIO/AI gains to achieve the different input limits. The last three examples in Table 20-1 illustrate this method. The last example shows a channel list with two modules.

**Table 20-1.** SCXI-1100 Channel Arrays, Input Limits Arrays, and Gains

| Array Index | SCXI-1100 Channel List Array | Input Limits Array | LabVIEW Selected SCXI Gain | LabVIEW Selected MIO/AI Gain |
|---|---|---|---|---|
| 0 | ob0!sc1!md1!0:7 | –0.01 to 0.01 | 1000 | 1 |
| 0 | ob0!sc1!md1!0:7 | –0.001 to 0.001 | 2000 | 5[1] |
| 0 | sc1!md1!0:7 | –0.001 to 0.001 | 2000 | 1 |
| 0<br>1 | ob0!sc1!md1!0:3<br>ob0!sc1!md1!4:15 | –0.1 to 0.1<br>–0.01 to 0.01 | 100<br>100 | 1<br>10 |
| 0<br>1 | ob0!sc1!md1!0:15<br>ob0!sc1!md1!16:31 | –0.01 to 0.01<br>–1.0 to 1.0 | 10<br>10 | 100[2]<br>1 |
| 0<br>1<br>2 | ob0!sc1!md1!0:3<br>ob0!sc1!md1!4:15<br>ob0!sc1!md2!0:7 | –1.0 to 1.0<br>–0.1 to 0.1<br>–0.01 to 0.01 | 10<br>10<br>1000 | 1<br>10<br>1 |

[1] Applies if the MIO/AI device supports a gain of 5 (some MIO/AI devices do not).

[2] This case forces a smaller gain at the SCXI module than at the MIO/AI device, because the input limits for the next channel range on the module require a small SCXI gain. This type of gain distribution is not recommended because it defeats the purpose of providing amplification for small signals at the SCXI module. The small input signals are only amplified by a factor of 10 before they are sent over the ribbon cable, where they are very susceptible to noise. To use the optimum gain distribution for each set of input signals, do not mix very small input signals with larger input signals on the same SCXI-1100 module, unless you are sampling them at different times.

You can open the AI Hardware Config VI to see the gain selection. After running this VI, the **group channel settings** cluster array at the right side of the panel shows the settings for each channel. The **gain** indicator displays the total gain for the channel, which is the product of the SCXI gain and the DAQ device gain, and the actual limit settings. The **group channel settings** cluster array also shows the input limits for each channel.

LabVIEW scales the input data as you specified, unless you select binary data only. Therefore, the gains are transparent to the application. You can specify the input signal limits and let LabVIEW do the rest.

## SCXI Settling Time

The filter and gain settings of your SCXI modules affect the settling time of the SCXI amplifiers and multiplexers. Always enter your jumpered filter settings and your jumpered gain settings (if applicable) in the Measurement & Automation Explorer or the configuration utility. LabVIEW uses the gain and filter settings to determine a safe interchannel delay that allows the SCXI amplifiers and multiplexers to settle between channel switching before sampling the next channel.

LabVIEW calculates the delay for you. If you set a scan rate that is too fast to allow for the default interchannel delay, LabVIEW shrinks the interchannel delay and returns a warning from the AI Start or AI Control VIs. Refer to your hardware manuals for SCXI settling times.

You can open the advanced-level AI Clock Config VI to retrieve the channel clock selection. Set the **which clock** control to **channel clock 1**, and set the **clock frequency** to `–1.00` (no change). Now run the VI. The **actual clock rate specification** cluster is on the right side of the panel.

**Note** When using NI 406*x* devices with SCXI, you cannot use the external triggering feature of the NI 406*x* device.

# 21

# Common SCXI Applications

Now that you have your SCXI system set up and you are aware of the special SCXI programming considerations, you should learn about some common SCXI applications. This section covers example VIs for analog input, analog output, and digital modules. For analog input, you will learn how to measure temperature (with thermocouples and RTDs) and strain (with strain gauges) using the SCXI-1100, SCXI-1101, SCXI-1102, SCXI-1112, SCXI-1121, SCXI-1122, SCXI-1125, SCXI-1127, SCXI-1141, SCXI-1142, and SCXI-1143 modules. If you are not measuring temperature or pressure, you can still gain basic conceptual information on how to measure voltages with an analog input module.

Another analog input module, the SCXI-1140, is a simultaneous sampling module. All the channels acquire voltages at the same time, which means you can preserve interchannel phase relationships. After all channel voltages are sampled by going into hold mode, the software will read one channel at a time. When a scan of channels is done, the SCXI-1140 module returns to track mode until the next scan period. Both of these operations are performed by the analog input VIs. You can use any of the DAQ VIs, located in the `labview\examples\daq\anlogin\anlogin.llb`, or the Getting Started Analog Input VI, found in `labview\examples\daq\run_me.llb`, to acquire data from the SCXI-1140 module.

For analog output, you will learn how to output voltage or current values using the SCXI-1124 module. For digital I/O, you will learn how to input values on the SCXI-1162/1162HV modules and output values on the SCXI-1160, SCXI-1161, and SCXI-1163/1163R modules.

# Analog Input Applications for Measuring Temperature and Pressure

Two common transducers for measuring temperature are thermocouples and RTDs. A common transducer for measuring pressure is strain gauges. Read the following sections on special measuring considerations needed for each transducer.

If you use the DAQ Channel Wizard to configure your analog input channels, you can simplify the programming needed to measure your channels. This section describes methods of measuring data using named channels configured in the DAQ Channel Wizard and using the conventional method.

**Note** For more information about using the DAQ Channel Wizard to configure your channels, refer to the *Configuring Your Channels in NI-DAQ 5.x, 6.x* section of Chapter 2, *Installing and Configuring Your Data Acquisition Hardware*. For more information about using channel names, refer to the *Channel Name Addressing* section of Chapter 3, *Basic LabVIEW Data Acquisition Concepts*.

## Measuring Temperature with Thermocouples

If you want to measure the temperature of the environment, you can use the temperature sensors in the terminal blocks. But if you want to measure the temperature of an object away from the SCXI chassis, you must use a transducer, like a thermocouple. A *thermocouple* is a junction of two dissimilar metals that gives varying voltages based on the temperature. However, when using thermocouples, you need to compensate for the thermocouple voltages produced at the screw terminal because the junction with the screw terminals itself forms another thermocouple. You can use the resulting voltage from the temperature sensor on the terminal block for cold-junction compensation. The cold-junction compensation voltage is used when linearizing voltage readings from thermocouples into temperature values.

The SCXI modules used to measure temperature in this section are the SCXI-1100, SCXI-1101, SCXI-1102, SCXI-1112, SCXI-1120, SCXI-1120D, SCXI-1121, SCXI-1122, SCXI-1125, SCXI-1127, SCXI-1141, SCXI-1142, and SCXI-1143. Most of the terminal blocks used with these modules have temperature sensors that can be used as cold-junction compensation.

In addition, the SCXI-1100, SCXI-1112, SCXI-1122, SCXI-1125, SCXI-1141, SCXI-1142, and SCXI-1143 offer a way for you to ground the module amplifier inputs so you can read the amplifier offset. You can subtract the amplifier offset value to determine the actual voltages.

## Temperature Sensors for Cold-Junction Compensation

The temperature sensors in the terminal blocks for the analog input modules can be used for cold-junction compensation. If you are operating your SCXI modules in multiplexed mode, leave the cold-junction sensor jumper on the terminal block in the `mtemp` (factory default) position. If you are using parallel mode, you can use the `dtemp` jumper setting.

**Note** The SCXI-1101, SCXI-1102, SCXI-1112, and SCXI-1127 use the `cjtemp` string only in multiplexed mode. The SCXI-1125 also uses the `cjtemp` string when the temperature sensor is configured in `mtemp` mode.

To read the temperature sensor, use the standard SCXI string syntax in the **channels** array with `mtemp` substituted for the channel number, as shown in the following table.

| Channel List Element | Channel Specified |
|---|---|
| `ob0!scx!mdy!mtemp` | The temperature sensor configured in `mtemp` mode on the multiplexed module in slot `y` of the chassis with ID `x`. |
| `ob0!scx!mdy!cjtemp` | The temperature sensor configured in `cjtemp` mode on the multiplexed SCXI-1102 module in slot `y` of the chassis with ID `x`, or the temperature sensor configured in `mtemp` mode on the multiplexed SCXI-1125 module in slot `y` of the chassis with ID `x`. |
| `ob0!scx!mdy!cjtempz` | The temperature sensor configured in `cjtemp` mode for the analog channel `z` on the multiplexed SCXI-1112 module in slot `y` of the chassis with ID `x`. |

If you want to read the cold-junction temperature sensor in `dtemp` mode, you can read the following onboard channels for these modules.

| Modules | Channel |
|---|---|
| SCXI-1100 | 1 |
| SCXI-1120/SCXI-1120D | 15<br>(use referenced single-ended mode) |
| SCXI-1121 | 4 |
| SCXI-1122 | 1 |

For example, you can run the Getting Started Analog Input VI, found in `labview\examples\daq\run_me.llb`, with the channel string `ob0!sc1!md1!mtemp` to read the temperature sensor on the terminal block connected to the module in slot 1 of SCXI chassis 1.

SCXI terminal blocks have two different kinds of sensors: an Integrated Circuit (IC) sensor or a thermistor. For terminal blocks that have IC sensors, such as the SCXI-1300 and the SCXI-1320, you can multiply the voltage read from the IC sensor by 100 to get the ambient temperature in degrees Centigrade at the terminal block. For terminal blocks that have thermistors, such as the SCXI-1303, SCXI-1322, SCXI-1327, and SCXI-1328, use the Thermistor Conversion VI, found in the **Functions»Data Acquisition»Signal Conditioning** palette, to convert the raw voltage data into units of temperature.

You cannot sample other SCXI channels from the same module while you are sampling the `mtemp` sensor. However, if you are in parallel mode, you can sample the `dtemp` sensor along with other channels on the same module at the same time because you are not performing any multiplexing on the SCXI module. You also can sample the `cjtemp` sensor along with other channels on the SCXI-1101, SCXI-1102, SCXI-1112, SCXI-1125, and SCXI-1127. For the SCXI-1102, `cjtemp` must be the first channel in the channel list. The SCXI-1112 and SCXI-1125 can sample `cjtemp` channels in any order in the channel list.

For greater accuracy, take several readings from the temperature sensor and average those readings to yield one value. If you do not want to average several readings, take a single reading using the Easy Analog Input VI, AI Sample Channel.

For more information, look at the SCXI Thermocouple example VIs, found in the appropriate module DLL in `labview\examples\daq\scxi\`. These VIs use the `cjtemp` or `mtemp` string to read the temperature sensor and use the reading for thermocouple cold-junction compensation.

## Amplifier Offset

The SCXI-1100, SCXI-1101, SCXI-1112, SCXI-1122, SCXI-1125, SCXI-1141, SCXI-1142, and SCXI-1143 have a special calibration feature that enables LabVIEW to ground the module amplifier inputs so that you can read the amplifier offset. For the other SCXI analog input modules, you must physically wire your terminals to ground. The measured amplifier offset is for the entire signal path including the SCXI module and the DAQ device.

To read the grounded amplifier on the SCXI-1100, SCXI-1101, or SCXI-1122, use the standard SCXI string syntax in the **channels** array with `calgnd` substituted for the channel number, as shown in the following table.

| Channel List Element | Channel Specified |
|---|---|
| `ob0!scx!mdy!calgnd` | (SCXI-1100 and SCXI-1122 only) The grounded amplifier of the module in slot `y` of the chassis with ID `x`. |
| `ob0!scx!mdy!calgndz` | Where `z` is the appropriate SCXI channel needing shunting for the SCXI-1112, SCXI-1125, SCXI-1141, SCXI-1142, or SCXI-1143. |

For example, you can run the Getting Started Analog Input VI, found in `labview\examples\daq\run_me.llb`, with the channel string `ob0!sc1!md1!calgnd` to read the grounded amplifier of the module in slot 1 of SCXI chassis 1. The voltage reading should be very close to 0 V. The AI Start VI grounds the amplifier before starting the acquisition, and the AI Clear VI removes the grounds from the amplifier after the acquisition completes.

The SCXI-1112, SCXI-1125, SCXI-1141, SCXI-1142, and SCXI-1143 have separate amplifiers for each channel, so you must specify the channel number when you ground the amplifier. To specify the channel number, attach the channel number to the end of the string `calgnd`. For example, `calgnd2` grounds the amplifier inputs for channel 2 and reads the offset.

You can also specify a range of channels. The string `calgnd0:7` grounds the amplifier inputs for channels 0 through 7 and reads the offset for each amplifier.

Use the Scaling Constant Tuner VI from **Functions»Data Acquisition» Signal Conditioning** to modify the scaling constants so LabVIEW automatically compensates for the amplifier offset when scaling binary data to voltage. The SCXI-1100 Voltage example, found in `labview\examples\daq\scxi\scxi1100.llb`, shows you a way to use the Scaling Constant Tuner VI.

## VI Examples

If you use the DAQ Channel Wizard to configure your channels, you can simplify the programming needed to measure your signal. LabVIEW configures the hardware with the appropriate input limits and gain, and performs cold-junction compensation, amplifier offsets, and scaling for you. To measure a channel using a channel name, you can use the Easy VIs or the Continuous Transducer VI located in `labview\examples\daq\solution\transduc.llb`, as shown in Figure 21-1. Enter the name of your configured channel in the **channels** input. The device input value is not used by LabVIEW when you use channel names. The acquired data is in the physical units you specified in the DAQ Channel Wizard.

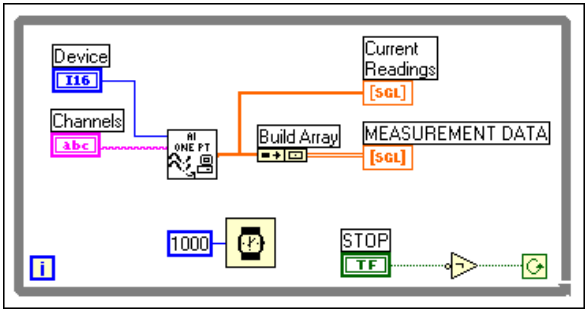



**Figure 21-1.** Continuous Transducer Measurement VI

The remainder of this section describes how to measure temperature with the SCXI-1100 and SCXI-112*x* modules using thermocouples when you do not use the DAQ Channel Wizard. The temperature examples below use both cold-junction measurements and amplifier offsets. In SCXI analog input examples, you cannot set the scaling constants with the Easy VIs (determined by the amplifier offset). With the Intermediate VIs, you can change the scaling constants before acquisition begins, and the Advanced VIs include functions that are not necessary to accurately

measure temperature with SCXI modules. The examples described in this section use Intermediate VIs along with transducer-specific VIs.

First, you should learn how to measure temperature using the SCXI-1100 with thermocouples. You can use the example SCXI-1100 Thermocouple VI located in `labview\examples\daq\scxi\scxi1100.llb`. Open the VI and continue reading this section.

To reduce the noise on the slowly varying signals produced by thermocouples, you can average the data and then linearize it. For greater accuracy, you can measure the amplifier offset, which helps scale the data and lets you eliminate the offset error from your measurement. The diagram below shows how you can program the Acquire and Average VI to measure the amplifier offset. You can find this VI in `vi.lib\daq\zdaqutil.llb`. This VI acquires 100 measurements from the amplifier offset, designated in the **offset channel** input by `calgnd`, and then averages the measurements. When you determine the amplifier offset, you must always use the same input limits and clock rates that you will be using in the acquisition. The Acquire and Average VI can measure the amplifier offset of many modules at once, but in Figure 21-2, it only measures one module.

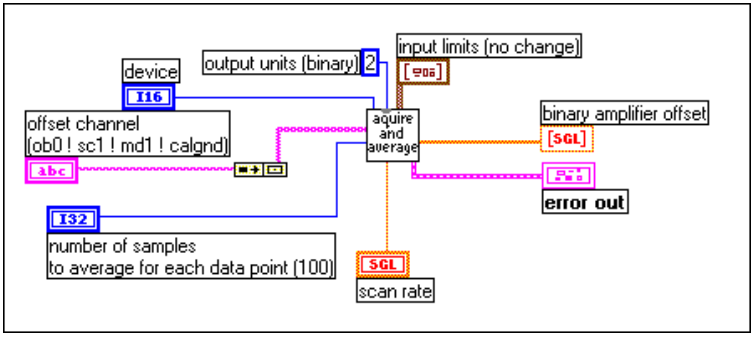



**Figure 21-2.** Measuring a Single Module with the Acquire and Average VI

After measuring the amplifier offset, measure the temperature sensor for cold-junction compensation. Both the amplifier offset and cold-junction measurements should be taken before any thermocouple measurements are taken. Use the Acquire and Average VI to measure temperature sensors, as shown in Figure 21-3. The main differences between the amplifier offset measurement and temperature sensor measurement are the channel string and the input limits. If you set the temperature sensor in `mtemp` mode (the most common mode), you access the temperature by using `mtemp`. If you set the temperature sensor in `dtemp` mode, you read the

corresponding DAQ device onboard channel. Make sure you use the temperature sensor input limits, which are different from your acquisition input limits. To read from a temperature sensor based on an IC sensor or a thermistor, set the input limit range from +2 to –2 V.

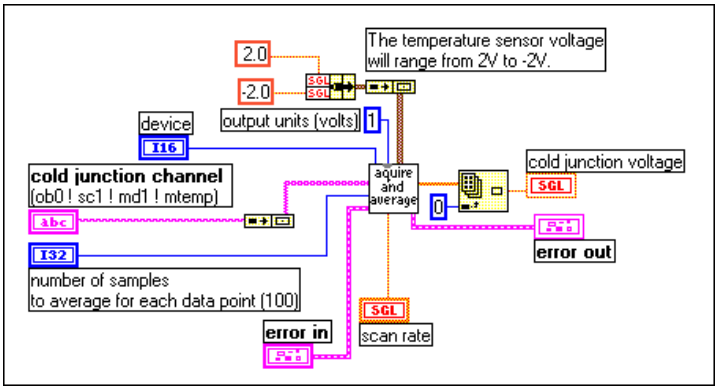



**Figure 21-3.** Measuring Temperature Sensors Using the Acquire and Average VI

After determining the average amplifier offset and cold-junction compensation, you can acquire data using the Intermediate VIs as shown in Figure 21-4. This example continually acquires data until an error occurs or the user stops the execution of the VI. In order to perform continuous, hardware-timed acquisition, you need to set up a buffer. In this case, the buffer is 10 times the number of points acquired for each channel. Before you initiate the acquisition with the AI Start VI, you need to set up the binary-to-voltage scaling constants by using the Scaling Constant Tuner VI. This VI, which you can find in **Functions»Data Acquisition»Signal Conditioning**, passes the amplifier offset to the DAQ driver so that LabVIEW accounts for the amplifier offset as the AI Read VI retrieves the data. After the compensated voltage data from the AI Read VI is averaged, the voltage values are converted to temperature and linearized by using the Convert Thermocouple Reading VI in **Functions»Data Acquisition»Signal Conditioning**. After completing the acquisition, remember to always clear the acquisition by using the AI Clear VI.

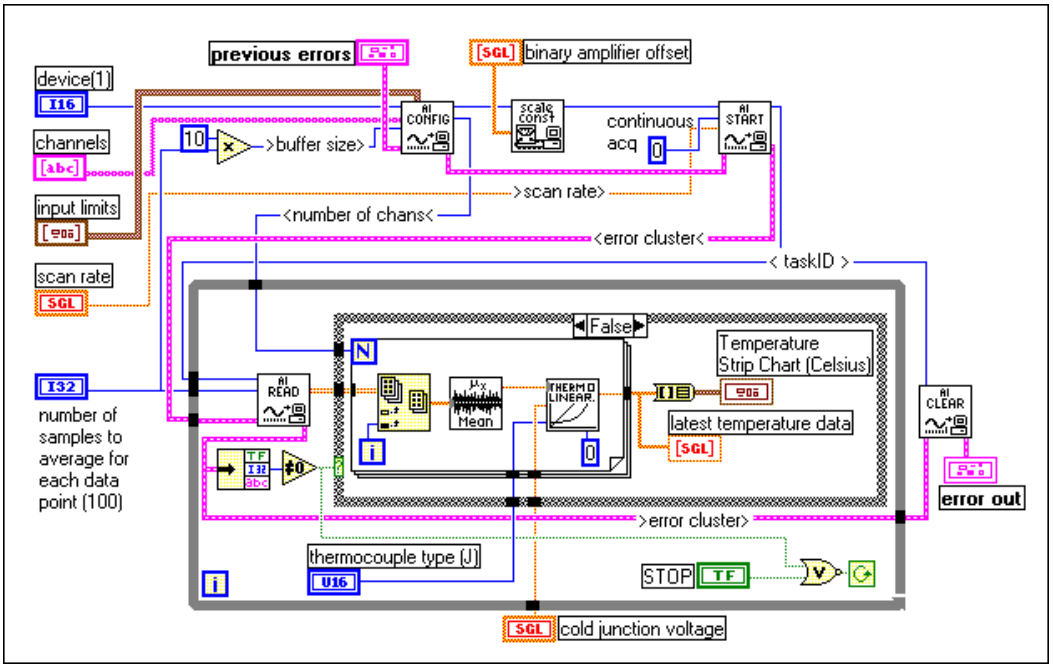



**Figure 21-4.** Continuously Acquiring Data Using Intermediate VIs

Another temperature acquisition example using the SCXI-1100 module is SCXI Temperature Monitor VI located in `labview\examples\daq\scxi\scxi_ai.llb`. This VI continually acquires thermocouple readings and sets an alarm if the temperature readings go above a user-defined limit.

You can use the SCXI-1100 examples with the SCXI-1122 module. Both modules have the capability to programmatically measure the amplifier offsets and both modules need the cold-junction compensation to linearize thermocouple measurements. The main differences between the two modules include the type of temperature sensors available on their terminal blocks and the way module channels are multiplexed.

The SCXI-1100 uses a CMOS multiplexer, which is capable of fast-channel multiplexing, whereas the SCXI-1122 uses an electromechanical relay to switch one of its 16 channels. Because the SCXI-1122 uses a relay, this module imposes a minimum interchannel delay of 10 ms, which affects the maximum possible scan rate. Scanning multiple SCXI-1122 channels many times can quickly wear out the relay. To avoid this, acquire data from the SCXI-1122 module a single channel at a time. For further information on reading SCXI-1122 channels, refer to

the *SCXI-1122 User Manual*, or the SCXI-1122 Voltage example VI in `labview\examples\daq\scxi\scxi1122.llb`.

If you are measuring temperature with the SCXI-1120 and SCXI-1121 modules, refer to the SCXI-1120/1121 Thermocouple example VI, located in `labview\examples\daq\scxi\scxi1120.llb` and `scxi1121.llb`. This VI is similar to the VI used to measure temperature on the SCXI-1100. Both VIs average and linearize temperature data using the Intermediate analog input VIs. The main differences between the VIs are that the SCXI-1120/1121 VI does not measure the amplifier offset, and the input limits for the module and the temperature sensor are different from the input limits for the SCXI-1100.

The SCXI-1120 and SCXI-1121 modules do not have the internal switch used to programmatically ground the amplifiers as in the SCXI-1100 for the amplifier offset measurement. To determine the amplifier offset, you must manually wire the amplifier terminals to ground and use a separate VI to read the offset voltage. You can also manually calibrate the SCXI-1120 and SCXI-1121 to remove any amplifier offset on a channel-by-channel basis. Refer to the SCXI-1120 or SCXI-1121 user manuals for specific instructions.

If you are measuring thermocouples with the SCXI-1125 or the SCXI-1112, refer to the examples in the appropriate LLB for the module located in `labview\examples\daq\scxi`. These examples demonstrate how to scan the cold-junction compensation channel (`cjtemp`) while scanning the thermocouple channels. By scanning the cold-junction sensor with the thermocouple channels, these examples are better suited to take temperature measurements over longer periods of time by accounting for temperature changes at the thermocouple junction inside the terminal block. The SCXI-1125 thermocouple example also demonstrates the ability to shunt the inputs and take an offset reading before collecting temperature data. This allows you to compensate for any offset drift due to operation at elevated temperatures or for offset produced by the system along the signal path.

## Measuring Temperature with RTDs

Resistance-temperature detectors (RTDs) are temperature-sensing devices whose resistance increase with temperature. They are known for their accuracy over a wide temperature range. RTDs require current excitation to produce a measurable voltage. RTDs are available in 2-wire, 3-wire, or 4-wire configuration. The lead wires in the 4-wire configuration are resistance-matched. If you use a 2-wire or 3-wire RTD, they are unmatched. Resistance in the lead wires that connect your RTD to the

measuring system will add error to your readings. If you are using lead lengths greater than 10 feet, you will need to compensate for this lead resistance. RTDs are also classified by the type of metal they use. The most common metal is platinum.

For more information about how the lead wires affect RTD measurements as well as general RTD information, refer to Application Note 046, *Measuring Temperature with RTDs—A Tutorial*. You can find this note on the National Instruments web site, `www.natinst.com`.

Signal conditioning is needed to interface an RTD to a DAQ device or an SCXI-1200 module. Signal conditioning required for RTDs include current excitation for the RTD, amplification of the measured signal, filtering of the signal to remove unwanted noise, and isolation of the RTD and monitored system from the host computer. Typically, you would use the SCXI-1121 module with RTDs because it easily performs all the signal conditioning listed previously. You must set up the excitation level, gain, and filter settings on the SCXI-1121 module with jumpers as well as in your system's configuration utility. For information on how to connect and configure the RTD with the SCXI-1121 module, look at the *Getting Started with SCXI* manual or the RTD application note mentioned previously.

The SC-2042 RTD is a signal conditioning device designed specifically for RTD measurement and can be used as an alternative to SCXI modules. For more information, refer to the National Instruments catalog.

You do not have to worry about cold-junction compensation with RTDs as you do when measuring thermocouples. To build an application in LabVIEW, you can use the Easy I/O analog input VIs. If you are measuring multiple transducers on several different channels, you will need to scan the necessary channels with little overhead. Because the Easy I/O VIs reconfigure your SCXI module every time your application performs an acquisition, it is recommended that you use the Intermediate analog input VIs.

Using the DAQ Channel Wizard to configure your channels can simplify the programming needed to measure your signal, as shown in Figure 21-5. LabVIEW configures the hardware with appropriate input limits and gain, measures the RTD, and scales the measurement for you. Enter the name of your configured channel in the **channels** input parameter. The acquired data is in the physical units you specify in the DAQ Channel Wizard.

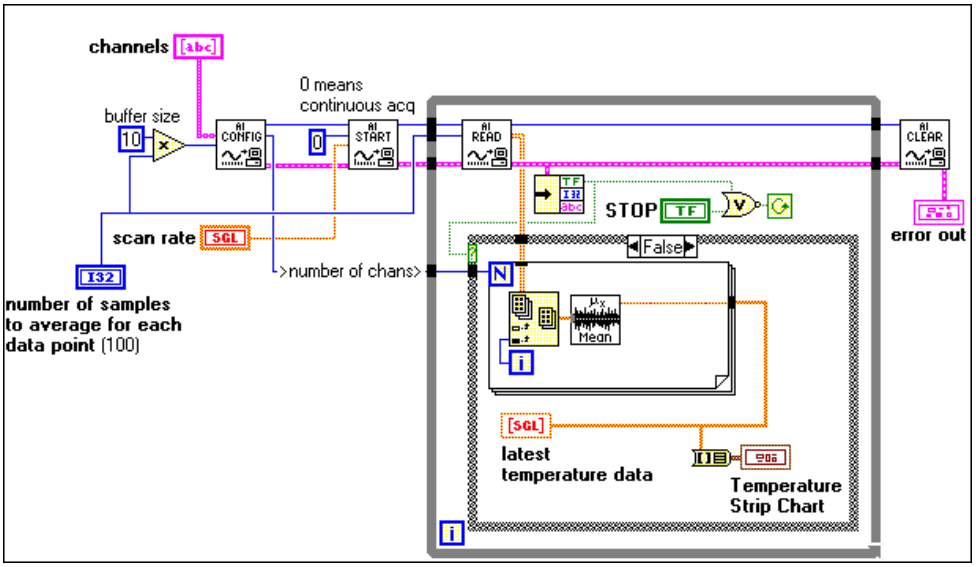



**Figure 21-5.** Measuring Temperature Using Information from the DAQ Channel Wizard

The VI in Figure 21-5 continually acquires data until an error occurs or you stop the VI from executing. To perform a continuous hardware-timed acquisition, you must set up a buffer. In this example, the buffer is 10 times the number of points acquired for each channel. For each acquisition, your device averages the temperature data. After completing the acquisition, always clear the acquisition by using the AI Clear VI.

If you are not using the DAQ Channel Wizard, you must use the RTD Conversion VI in addition to specifying additional input parameters, as shown in Figure 21-6. The Convert RTD Reading VI, in **Functions»Data Acquisition»Signal Conditioning**, converts the voltage read from the RTD to a temperature representation.

**Note** You should only use the RTD conversion function in LabVIEW for platinum RTDs. If you do not have a platinum RTD, the voltage-temperature relation will be different, so the LabVIEW conversion function cannot be used.

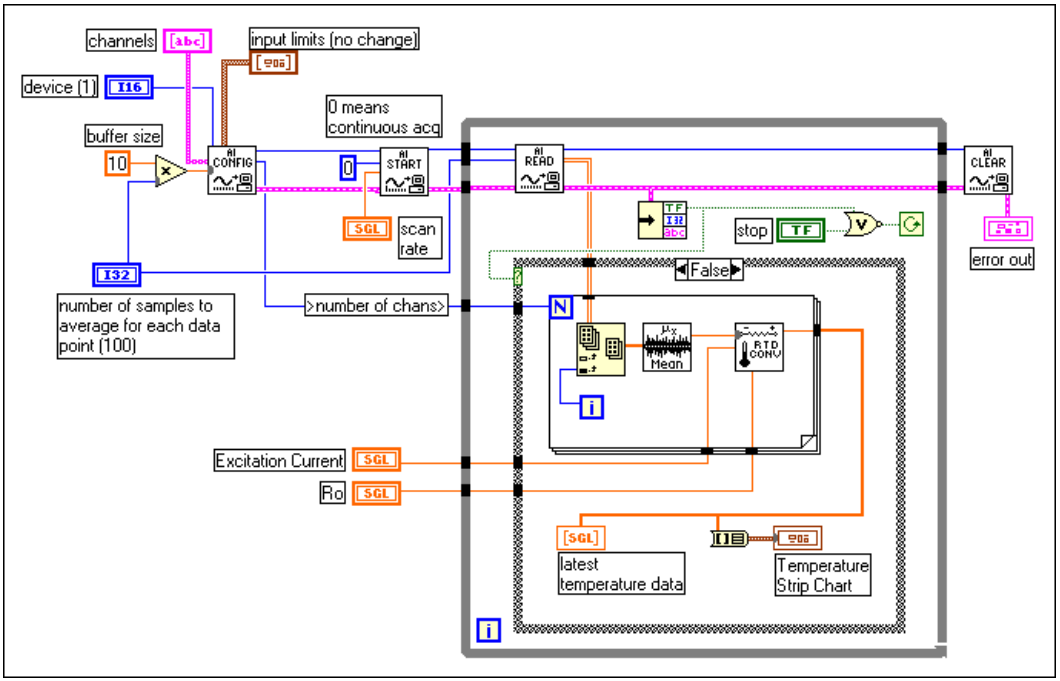



**Figure 21-6.** Measuring Temperature Using the Convert RTD Reading VI

The VI in Figure 21-6 continually acquires data until an error occurs or you stop the VI from executing. In order to perform a continuous hardware-timed acquisition, you need to set up a buffer. In this example, the buffer is 10 times the number of points acquired for each channel. After your device averages the voltage data from the AI Read VI, it converts the voltage values to temperature. After completing the acquisition, remember to always clear the acquisition by using the AI Clear VI.

## Measuring Pressure with Strain Gauges

Strain gauges give varying voltages in response to stress or vibrations in materials. Strain gauges are thin conductors attached to the material to be stressed. Resistance changes in parts of the strain gauge to indicate deformation of the material. Strain gauges require excitation (generally voltage excitation) and linearization of their voltage measurements. Depending on the strain gauge configuration, another requirement for using strain gauges with SCXI is a configuration of resistors. As shown in Figure 21-7, the resistance from the strain gauges combined with the SCXI hardware form a diamond-shaped configuration of resistors, known as a *Wheatstone bridge*. When you apply a voltage to the bridge, the

differential voltage ($V_m$) varies as the resistor values in the bridge change. The strain gauge usually supplies the resistors that change value with strain.

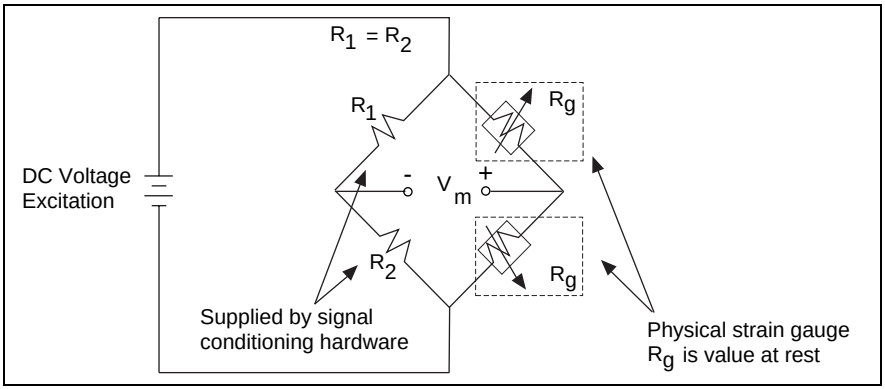



**Figure 21-7.** Half-Bridge Strain Gauge

Strain gauges come in full-bridge, half-bridge, and quarter-bridge configurations. For a full-bridge strain gauge, the four resistors of the Wheatstone bridge are physically located on the strain gauge itself. For a half-bridge strain gauge, the strain gauge supplies two resistors for the Wheatstone bridge while the SCXI module supplies the other two resistors, as shown above. For a quarter-bridge strain gauge, the strain gauge only supplies one of the four resistors for a Wheatstone bridge. For more information on how to connect your strain gauge to SCXI, refer to the *Getting Started with SCXI* manual.

The SCXI-1121 and the SCXI-1122 modules are commonly used with strain gauges because they include voltage or current excitation and internal Wheatstone bridge completion circuits. You can also use the signal conditioning device SC-2043SG as an alternative to SCXI modules. The device is designed specifically for strain gauge measurements. For more information on this device, refer to your National Instruments catalog.

You can set up your SCXI module to amplify strain gauge signals or filter noise from signals. In order to set up the excitation level, gain, and filter settings, consult your *Getting Started with SCXI* manual for the necessary hardware configuration and Chapter 2, *Installing and Configuring Your Data Acquisition Hardware*, for software configuration.

To build a strain gauge application in LabVIEW, you can use the Easy I/O analog input VIs. If you are measuring multiple transducers on several different channels, you need to scan the necessary channels as quickly as possible. Because the Easy I/O VIs reconfigure your SCXI module every time the VI is called, you should use the Intermediate analog input VIs as well as the Strain Gauge Conversion VI, as shown in Figure 21-8. The Convert Strain Gauge Reading VI, located in **Functions»DAQ»DAQ Utilities**, converts the voltage read by the strain gauge to units of strain.

Using the DAQ Channel Wizard to configure your channels simplifies the programming required to measure your signal, as shown in Figure 21-8. LabVIEW configures the hardware with the appropriate input limits and gain, measures the strain gauge, and scales the measurement for you. Enter the name of your configured channel in the **channels** input. You do not need to wire the **device** or **input limits** input. The acquired data is in the physical units you specified in the DAQ Channel Wizard.

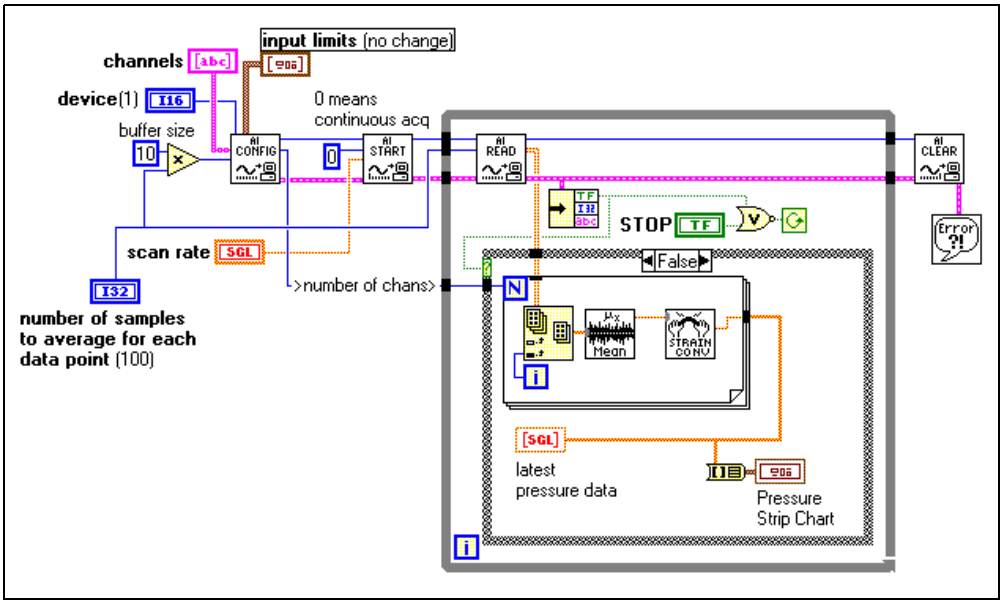



**Figure 21-8.** Measuring Pressure Using Information from the DAQ Channel Wizard

Figure 21-8 continually acquires data until an error occurs or you stop the VI from executing. In order to perform continuous acquisition, you need to set up a buffer. In this example, the buffer is 10 times the number of points acquired for each channel. After your device averages the voltage data from the AI Read VI, it converts the voltage values to strain values. After

completing the acquisition, remember to always clear the acquisition by using the AI Clear VI.

When measuring strain gauge data, there are some parameters on the Convert Strain Gauge Reading VI, shown in Figure 21-9, you should know.

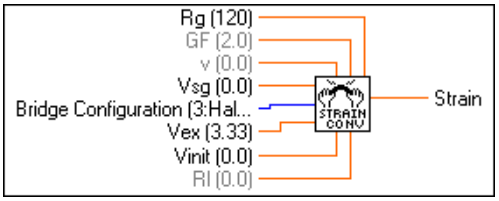



**Figure 21-9.** Convert Strain Gauge Reading VI

**Vsg**, the strain gauge value, is the only parameter wired in the previous VI diagram. The other parameters for this VI have default values but those values may not be correct for your strain gauge. You should check the following parameters:

- **Rg**—The resistance of the strain gauge before strain is applied. You can usually ignore the lead resistance
- **Bridge Configuration**
- **Vex**—The excitation voltage
- **Vinit**—The voltage across the strain gauge before strain is applied (always measure at the beginning of the VI)
- **Rl**—The lead resistance
- **Rl**—For strain gauges unless the leads are several feet

For more information on any of the parameters for this VI, refer to Chapter 30, *Signal Conditioning VIs*, in the *LabVIEW Function and VI Reference Manual*, or refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**.

# Analog Output Application Example

You can output isolated analog signals using the SCXI-1124 analog output module. If you use the DAQ Channel Wizard to configure your analog output channels, generating signals using the SCXI-1124 is no different from the techniques in Part III, *Making Waves with Analog Output*. The remainder of this section describes how to generate signals with the SCXI-1124 when you do not use the DAQ Channel Wizard.

The SCXI-1124 can generate voltage and current signals. Refer to the example analog output VI, SCXI-1124 Update Channels VI, located in `labview\examples\daq\scxi\scxi1124.llb`. This VI uses the analog output Advanced VIs because the output mode (whether you have voltage or current data) must be accessible in order to change the value, as shown in Figure 21-10. The program calls the AO Group Config VI to specify the device and output channels. The AO Hardware Config VI specifies the output mode and the output range, or limit settings, for all the channels specified in the channels string. This advanced-level VI is the only place where you can specify a voltage or current output mode. If you are going to output voltages only, you may want to use the AO Config VI (an Intermediate VI), instead of the AO Group Config and AO Hardware Config VIs. You can program individual output channels of the SCXI-1124 for different output ranges by using the arrays for channels, output mode, and limit settings. The AO Single Update VI initiates the update of the SCXI-1124 output channels. To help debug your VIs, it is always helpful to display any errors, in this case using the Simple Error Handler VI.

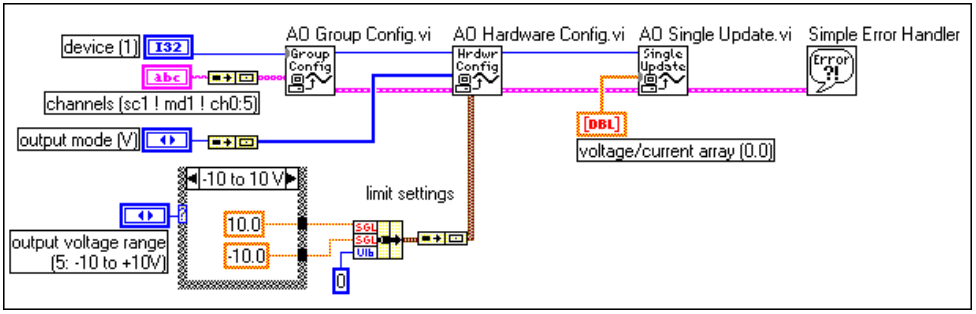



**Figure 21-10.** SCXI-1124 Update Channels VI

# Digital Input Application Example

To input digital signals through an SCXI chassis, you can use the SCXI-1162 and SCXI-1162HV modules and the Easy Digital VI, Read from Digital Port, as shown in Figure 21-11.

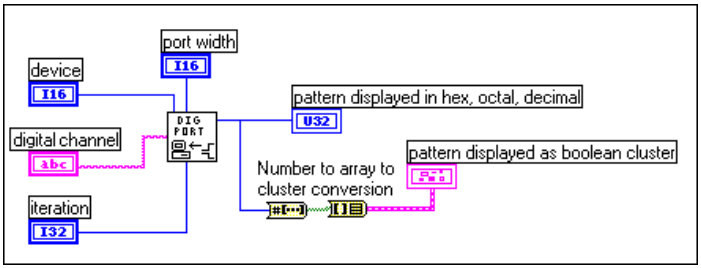



**Figure 21-11.** Inputting Digital Signals through an SCXI Chassis Using Easy Digital VIs

If you configure channels using the DAQ Channel Wizard, **digital channel** can consist of a digital channel name. The channel name can refer to either a port or a line in a port. You do not need to specify **device**, **line**, or **port width**, as these inputs are not used by LabVIEW if a channel name is specified in **digital channel**.

As an alternative, **digital channel** can be expressed in the `scx!mdy!0` format, where you are trying to input from the digital input module on slot `y` of chassis `x`. The last identifier is always port 0, because the whole module is considered one port. In this example, you must also specify **device** and **port width**. The **port width** should be the number of lines in a port on your SCXI module if you are operating in multiplexed mode. For the SCXI-1162 and SCXI-1162HV, the **port width** is 32 lines. If you are operating in parallel mode, the **port width** should be the number of lines on your DAQ device. The DIO-32F device can access all 32 lines of the SCXI modules at once by using the SCXI-1348 cable assembly. The DIO-24 and the DIO-96 devices can only access the first 24 lines of these modules when configured in parallel mode. For the fastest performance in parallel mode, you can use the appropriate onboard port numbers instead of the SCXI channel string syntax.

Use the iteration input to optimize your digital operation. When **iteration** is `0` (default), LabVIEW calls the DIO Port Config VI (an Advanced VI) to configure the port. If **iteration** is greater than zero, LabVIEW bypasses reconfiguration and remembers the last configuration, which improves performance. You can wire this input to an iteration terminal of a loop. With the DIO-24 and DIO-96 devices, every time you call the DIO Port

Config VI, the digital line values are reset to default values. If you want to maintain the integrity of the digital values from one loop iteration to another, do not set **iteration** to `0` except for the first iteration of the loop.

For an example on SCXI digital input, refer to SCXI-1162/1162HV Digital Input VI located in `labview\examples\daq\scxi\scxi1162.llb`. Even though this VI uses Advanced VIs, it is functionally equivalent to the Easy I/O Digital VI, Read from Digital Port.

**Note** The DIO Port Config VI resets output lines on adjacent ports on the same 8255 chip for DIO-24, DIO-96, AT-MIO-16D, AT-MIO-16DE, and Lab and 1200 Series devices.

**Note** If you are also using SCXI analog input modules, make sure your cabling DAQ device is cabled to one of them.

# Digital Output Application Example

To output digital signals through an SCXI chassis, you can use the SCXI-1160, SCXI-1161, SCXI-1163, and SCXI-1163R modules and the digital Easy Digital VI, Write to Digital Port, as shown in Figure 21-12.

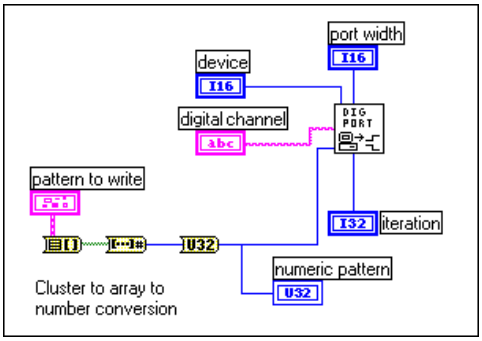



**Figure 21-12.** Outputting Digital Signals through an SCXI Chassis Using Easy Digital VIs

If you configure channels using the DAQ Channel Wizard, **digital channel** can consist of a digital channel name. The channel name can refer to either a port or a line in a port. You do not need to specify **device**, **line**, or **port width**, as theses inputs are not used by LabVIEW if a channel name is specified in **digital channel**.

As an alternative, **digital channel** can be expressed in the `scx!mdy!0` format, where you are trying to output from the digital output module on

slot y of chassis x. The last identifier is always port 0, because the whole module is considered one port. In this case, you must also specify **device** and **port width**. The **port width** should be the number of lines on your SCXI module if you are operating in multiplexed mode. The SCXI-1160 has 16 relays, the SCXI-1161 has eight relays, and the SCXI-1163/1163R have 32 relays. You can not use the SCXI-1160 or SCXI-1161 in parallel mode. For the SCXI-1163/1163R the **port width** in parallel mode should be the number of lines on your DAQ device or SCXI-1200 module. The DIO-32F device can access all 32 lines of the SCXI-1163/1163R modules at once by using the SCXI-1348 cable assembly. The DIO-24 and the DIO-96 devices can only access the first 24 lines of the SCXI-1163/1163R when configured in parallel mode. For the fastest performance in parallel mode, you can use the appropriate onboard port numbers instead of the SCXI channel string syntax.

Use the **iteration** input to optimize your digital operation. When **iteration** is `0` (default), LabVIEW calls the DIO Port Config VI (an Advanced VI) to configure the port. If **iteration** is greater than zero, LabVIEW bypasses reconfiguration and remembers the last configuration, which improves performance. You can wire this input to an iteration terminal of a loop. Every time you call the DIO Port Config VI the digital line values are reset to default values. If you want to maintain the integrity of the digital values from one loop iteration to another, do not set iteration to `0` except for the first iteration of the loop.

For an example on SCXI digital output, refer to SCXI-116x Digital Output VI located in `labview\examples\daq\scxi\scxi1160.llb`, `scxi1161.llb`, or `scxi1163.llb`. Even though this VI uses Advanced VIs, it is functionally equivalent to the Easy Digital VI, Write to Digital Port.

**Note** If you also are using SCXI analog input modules, make sure your cabling DAQ device is cabled to one of them.

# Multi-Chassis Applications

Multiple SCXI-1000, SCXI-1000DC, or SCXI-1001 chassis can be daisy-chained using the SCXI-1350 or SCXI-1346 multichassis cable adapters and an MIO Series DAQ device other than the DAQPad-MIO-16XE-50. Every module in each of the chassis must be in multiplexed mode. Only one of the chassis will be connected directly to the DAQ device. Also, if you are using Remote SCXI with RS-485, you can daisy chain up to 31 chassis on a single RS-485 port. Because you can only configure up to 16 devices on the NI-DAQ Configuration Utility, you can only have up to 16 SCXI-1200s in your system.

**Note** Lab Series devices, LPM devices, DAQCard-500, 516 devices, DAQCard-700, 1200 Series (other than SCXI-1200), and DIO-24 devices do not support multi-chassis applications.

If you use the DAQ Channel Wizard to configure your analog input channels, you simply address channels in multiple chassis by their channel names. Channel names can be combined, separated by commas, to measure data from multiple modules in a daisy chain configuration at the same time. For example, if you have a named channel called `temperature` on one module in the daisy chain and `pressure` on another module in the same daisy chain, your **channels** array could be `temperature, pressure`. You must enter the chassis in a sequential order in the NI-DAQ Configuration Utility, assigning the first chassis in the chain an ID number of 1, the second chassis an ID number of 2, and so forth.

If you are not using the DAQ Channel Wizard, there are special considerations for addressing the channels. When you daisy chain multiple chassis to a single DAQ device (non-Remote SCXI), each chassis multiplexes all of its analog input channels into a separate onboard analog input channel. The first chassis in the chain uses onboard channel 0, the second chassis in the chain uses onboard channel 1, and so on. To access channels in the second chassis, you must select the correct onboard channel as well as the correct chassis ID. The string `ob1!sc2!md1!0` means channel 0 on the module in slot 1 of SCXI chassis 2, multiplexed into onboard channel 1. Remember to use the correct chassis ID number from the configuration utility and to put the jumpers from the power supply module in the correct position for each chassis.

When an MIO/AI Series device is cabled by a ribbon cable or shielded cable to multiple chassis, the number of reserved analog input channels depends on the number of chassis. On MIO Series devices, lines 0, 1, and 2 are unavailable. On MIO-E Series devices, lines 0, 1, 2, and 4 are unavailable. For more channel information refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**.

When you access digital SCXI modules, you do not use onboard channels. Therefore, if you have multiple chassis, you only have to choose the correct SCXI chassis ID and module slot.

When you use Remote SCXI to address analog input channels, specify the device number of the SCXI-1200 that is located in the same chassis containing the analog input module from which you take samples.

You can perform DAQ operations on channels in multiple SCXI chassis at the same time. For example, the first element of your **channels** array could be `ob0!sc1!md1!0:31`, and the second element of the **channels** array could be `ob1!sc2!md1!0:31`. Then, LabVIEW would scan 32 channels on module 1 of SCXI chassis 1, using onboard channel 0, then the 32 channels on module 1 in SCXI chassis 2, using onboard channel 1. Remember that the **scan rate** you specify is how many scans per second LabVIEW performs. For each scan, LabVIEW reads every channel in the **channels** array.

You can practice reading channels from different chassis by using the channel strings explained above in the Easy VIs.

# 22

# SCXI Calibration—Increasing Signal Measurement Precision

Your SCXI module ships to you factory-calibrated for the specified accuracy. You need to recalibrate the module only if the precision of your signal measurement is not acceptable because of shifts in environmental conditions.

**Note** This chapter does not apply to the SCXI-1200. For calibration on the SCXI-1200, you should use the 1200 Calibrate VI, which you can find in **Functions»Data Acquisition»Calibration and Configuration**. If you are using an SCXI-1200 in a remote SCXI configuration, National Instruments recommends that you connect directly to your parallel port to perform calibration, because it works much faster.

## EEPROM—Your System's Holding Tank for Calibration Constants

When you calibrate your SCXI module in LabVIEW, the calibration constants can be stored in *electronically erasable programmable read-only memory (EEPROM)*. The EEPROM stores calibration constant information in your module's memory. There are three parts to the EEPROM: the factory area, the default load area, and the user area.

**Note** Only the SCXI-1102, SCXI-1112, SCXI-1122, SCXI-1124, SCXI-1125, SCXI-1141, SCXI-1142, and SCXI-1143 have EEPROMs. All other SCXI modules do not store calibration constants.

**Note** The SCXI-1125 does not have a user area in its EEPROM.

- The *factory area* has a set of factory calibration constants already stored in it when you receive your SCXI module. You cannot write into the factory area, but you can read from it, so you can always access and use these factory constants if they are appropriate for your application.
- The *default load area* is where LabVIEW automatically looks to load calibration constants the first time you access the module. When the module is shipped, the default load area contains a copy of the factory calibration constants.

**Note** You may overwrite the constants stored in the default load area of EEPROM with a new set of constants using the SCXI Cal Constants VI. To learn more about this VI, refer to Chapter 29, *Calibration and Configuration VIs*, in the *LabVIEW Function and VI Reference Manual*, or refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**.

- The *user area* is an area for you to store your own calibration constants that you calculate using the SCXI Cal Constants VI. You can also put a copy of your own constants in the default load area if you want LabVIEW to automatically load your constants for subsequent operations. You can read and write to the user area.

**Note** Use the user area in EEPROM to store any calibration constants that you may need to use later. This safeguards you from accidentally overwriting your constants in the default load area, because you will have two copies of your new constants and you can revert to the factory constants by copying the factory area to the default load area without wiping out your new constants entirely.

# Calibrating SCXI Modules

The SCXI Cal Constants VI in LabVIEW automatically calculates the calibration constants for your module with the precision you need for your particular application. You can find this VI in **Functions»DAQ» Calibration and Configuration**. Refer to Chapter 29, *Calibration and Configuration VIs*, in the *LabVIEW Function and VI Reference Manual* for specifics on the SCXI Cal Constants VI and each of its parameters, or refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**.

For the SCXI-1112 and SCXI-1125 modules, the SCXI Calibrate VI can be used for easy one-point calibration (SCXI-1125) or two-point calibration (SCXI-1112) without the separate function calls necessary with the SCXI Cal Constants VI. One- and two-point calibration is described in the *SCXI Calibration Methods for Signal Acquisition* section later in this chapter. The SCXI Calibrate VI also can be found in **Functions»DAQ»Calibration and Configuration**.

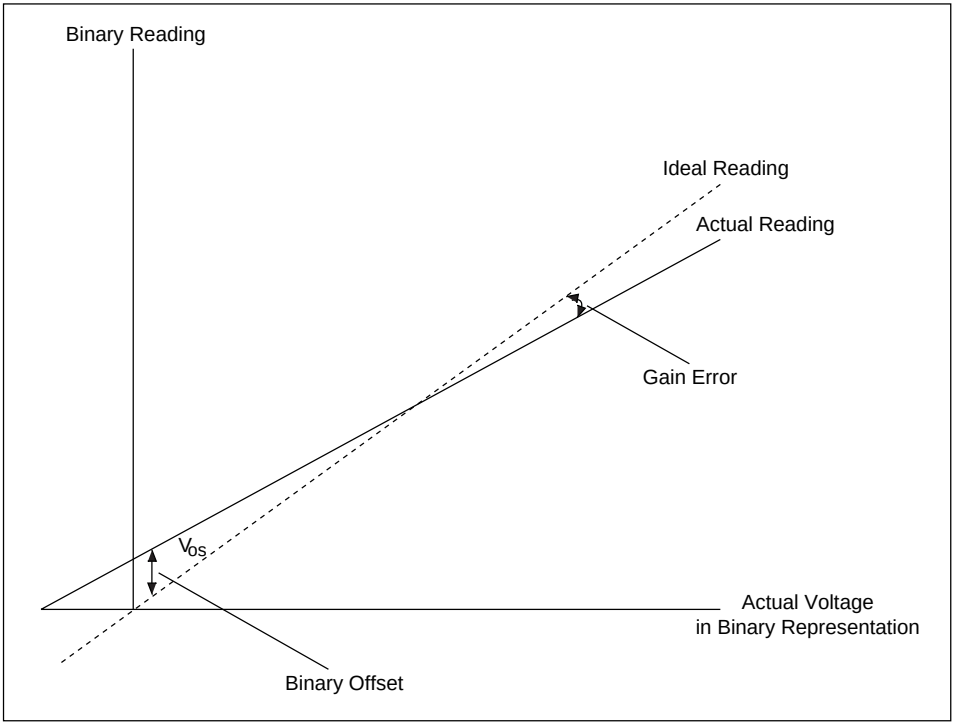
By default, calibration constants for the SCXI-1102, SCXI-1112, SCXI-1122, SCXI-1125, SCXI-1141, SCXI-1142, and SCXI-1143 are loaded from the module EEPROM. The SCXI-1141, SCXI-1142, and SCXI-1143 have only gain adjust constants in the EEPROM. They do not have the binary zero offset. All other analog input modules do not have calibration constants by default and do not assume any binary offset and ideal gain settings. This means you must use one of the procedures described in the *SCXI Calibration Methods for Signal Acquisition* section to store calibration constants for your module if it is not an SCXI-1102, SCXI-1112, SCXI-1122, SCXI-1125, SCXI-1141, SCXI-1142, or SCXI-1143.

You can determine calibration constants based on your application setup, which includes your type of DAQ device, DAQ device settings, and cable assembly—combined with your SCXI module and its configuration settings.

**Note** If your SCXI module has independent gains on each channel, the calibration constants for each channel are stored at each gain setting.

## SCXI Calibration Methods for Signal Acquisition

There are two ways you can calibrate your SCXI module—through *one-point calibration* or *two-point calibration*. The following illustration explains why you may need to calibrate your SCXI module.




This picture shows the difference between the ideal reading and the actual reading. This difference is called $V_{os}$, or the **binary offset**, before the two readings intersect. The difference in slope between the actual and ideal readings is called the **gain error**.

One-point calibration removes the $V_{os}$ (**binary offset**) by measuring a 0 V signal and comparing the actual reading to it. Two-point calibration removes the $V_{os}$ (**binary offset**) *and* corrects gain error by first performing a one-point calibration. Then you measure a voltage at *x* volts and compare it to the actual reading. The *x* must be as close as possible to the full-scale range. The following sections explain how to perform a one-point and two-point calibration.

## One-Point Calibration

You should use one-point calibration when you only need to adjust the binary offset in your module. If you need to adjust both the binary offset and the gain error of your module, read the *Two-Point Calibration* section later in this chapter.

**Note** If you are using an AT-MIO-16F-5, AT-MIO-64F-5, AT-MIO-16X device or an E Series device, you should calibrate your DAQ device first using either the MIO Calibrate VI or E-Series Calibrate VI.

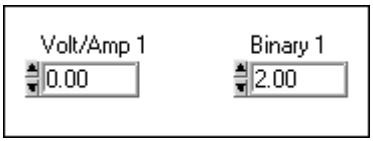
To perform a one-point calibration calculation in LabVIEW, complete the following steps:

1. Make sure you set the SCXI gain to the gain you want to use in your application. If your modules have gain jumpers or DIP switches, they must be set appropriately. Refer to your SCXI module user manual for jumper or switch setting information. If your modules have software-programmable gain, use the **input limits** parameter in the AI Config VI to set gain.

2. Program the module for a single-channel operation by using the AI Config VI with the channel that you are calibrating as the **channels** parameter in the VI.
3. Ground your SCXI input channel to determine the binary zero offset. You should ground inputs because offset can vary at different voltage levels due to gain error. If you are using an SCXI-1100 or SCXI-1122, you can ground your input channels without external hookups by substituting the channel string with `calgnd` as the channel number. For other modules, you need to wire the positive and negative channel inputs together at the terminal block and wire them to the chassis ground.

4. Use the AI Single Scan VI to take several readings and average them for greater accuracy. Set the DAQ device gain settings to match the settings you plan to use in your application. If you are using an AT-MIO-16F-5, AT-MIO-64F-5, or AT-MIO-16X, use the MIO Configure VI to enable dithering, which makes your averaged data more accurate. The dither mode is always enabled on E-series devices. By using the AI Start and AI Read VIs, instead of the AI Single Scan VI, you can average over an integral number of 60 Hz or 50 Hz power line cycles (sine waves) to eliminate line noise. You now have your first volt/binary measurement: volt = 0.0 or the applied voltage at your input channel, and binary is your binary reading or binary average.

5. Use the SCXI Cal Constants VI with your volt/binary measurement from step 4 as the **Volt/Amp/Hz 1** and **Binary 1** inputs in your VI, respectively. (These input names may vary depending on your application setup.) For example, if your volt/binary measurement from step 4 was 0.00 V and 2, then you would enter the values into your front panel controls as shown in the following illustration.



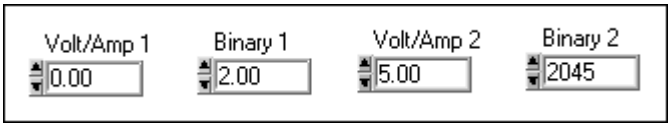

## Two-Point Calibration

The following steps show you how to perform a two-point calibration calculation in LabVIEW. You should use two-point calibration when you need to correct both the binary offset and the gain error in your SCXI module.

**Note** If you are using an AT-MIO-16F-5, AT-MIO-64F-5, or AT-MIO-16X device or an E Series device, you should calibrate your DAQ device first using either the MIO Calibrate VI or E-Series Calibrate VI.

To perform a two-point calibration calculation in LabVIEW, follow steps 1 through 5 in the previous section, *One-Point Calibration*, then complete the following steps:

6. Now apply a known, stable, non-zero voltage to your input channel at the terminal block. This input voltage should be close to the upper limit of your input voltage range for the given gain setting. For example, if your input voltage range is –5 to 5 V, you would want to apply an input voltage that is as close to 5 V as possible, but does not exceed 5 V.
7. Take another binary reading or average of readings. If your binary reading is the maximum binary reading for your DAQ device, try a smaller input voltage. This is your second volt/binary measurement.
8. Use the SCXI Cal Constants VI with the first volt/binary measurement from step 4 as **Volt/Amp/Hz 1** and **Binary 1** inputs, and the second measurement from step 7 as **Volt/Amp/Hz 2** and **Binary 2** inputs of the VI. The following illustration shows how you should enter the values into these inputs in LabVIEW if your volt/binary measurements are 0 V/0 and 5 V/2045. Your input names may vary depending on your application setup.




9. If you are using SCXI-1102 or SCXI-1122 inputs, you can save the constants in the module user area in EEPROM. Store constants in the user area as you are calibrating, and then use SCXI Cal Constants VI again at the end of your calibration sequence to copy the calibration table in the user area to the default load area in EEPROM. Remember, constants stored in the default load area can be overwritten. If you want to use a set of constants later, keep a copy of the constants stored in the user area in EEPROM.

**Note** If you are storing calibration constants in the SCXI-1102 or SCXI-1122 EEPROM, your binary offset and gain adjust factors must not exceed the ranges given in the respective module user manuals.

For other analog input modules, you must store the constants in the memory. Unfortunately, calibration constants stored in the memory are lost at the end of a program session. You can solve this problem by creating a file and saving the calibration constants to this file. You can load them again in subsequent application runs by passing them into the SCXI Cal Constants or the Scale Constant Tuner VIs.

## Calibrating SCXI Modules for Signal Generation

When you output a voltage or current value to your SCXI analog output module, LabVIEW uses the calibration constants loaded for the given module, channel, and output range to scale the voltage or current value to the appropriate binary value to write to the output channel. By default, calibration constants for the SCXI-1124 are loaded into the memory from the EEPROM default load area.

Recalibrate your SCXI analog output module by following these steps:

1. Use the AO Single Update VI to output a binary value. If you are calibrating a voltage output range, enter `0` in the **binary array** input of the VI. If you are calibrating current range, enter `255` into the **binary array** input of the VI.
2. Measure the output voltage or current at the output channel with a voltmeter or ammeter. This is your first volt/binary measurement: **Binary 1** = 0, and **Volt/Amp/Hz 1** is the voltage or current you measured at the output.

3. Use the AO Single Update VI to output a binary value of 4,095.
4. Measure the output voltage or current at the output channel. This is your second volt/binary measurement: **Binary 2** should be `4,095` and **Volt/Amp 2** is the voltage or current you measured at the output.
5. Use SCXI Cal Constants VI with the first voltage/binary measurement from step 2 as the **Volt/Amp/Hz 1** and **Binary 1** inputs and the second measurement from step 4 as the **Volt/Amp/Hz 2** and **Binary 2** inputs of the VI.

You can save the constants on the module in the user area in EEPROM. Use the user area as you are calibrating, and then use SCXI Cal Constants VI again at the end of your calibration sequence to copy the calibration table in the user area to the default load area in EEPROM. Remember, constants stored in the default load area can be overwritten. If you want to use a set of constants later, keep a copy of the constants stored in the user area in EEPROM.

Repeat the procedure above for each channel and range you want to calibrate. Subsequent analog outputs will use your new constants to scale voltage or current to the correct binary value.

For more information on the SCXI Cal Constants VI, refer to Chapter 29, *Calibration and Configuration VIs*, in the *LabVIEW Function and VI Reference Manual*, or refer to the LabVIEW *Online Reference*, by selecting **Help»Online Reference**.

# Part VI

# Counting Your Way to High-Precision Timing

This section describes the different ways you can use counters with your data acquisition application, including generating a pulse or pulses; measuring pulse width, frequency, and period; counting events and time; and dividing frequencies for precision timing.

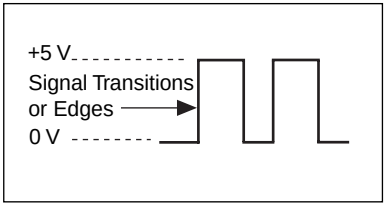
Part VI, *Counting Your Way to High-Precision Timing*, contains the following chapters:

- Chapter 23, *Things You Should Know about Counters*, shows you how to add high-precision timing to your DAQ system by using counters and explains basic counter concepts.
- Chapter 24, *Generating a Square Pulse or Pulse Trains*, describes the ways you can generate a square pulse or multiple pulses (called *pulse trains*) using the counters available on your DAQ device with the Easy, Intermediate, and Advanced Counter VIs in LabVIEW.
- Chapter 25, *Measuring Pulse Width*, describes how you can use a counter to measure pulse width.
- Chapter 26, *Measuring Frequency and Period*, describes the various ways you can measure frequencies and periods using the counters on your DAQ device.
- Chapter 27, *Counting Signal Highs and Lows*, teaches you how to use counters to count external events or elapsed time.
- Chapter 28, *Dividing Frequencies*, shows you how to divide the available device frequencies to get the frequency you need for your data acquisition application.

# 23

# Things You Should Know about Counters

Counters add counting or high-precision timing to your DAQ system. Counters respond to and output TTL signals—square-pulse signals that are 0 V (low) or 5 V (high) in value. The following diagram shows a TTL signal.

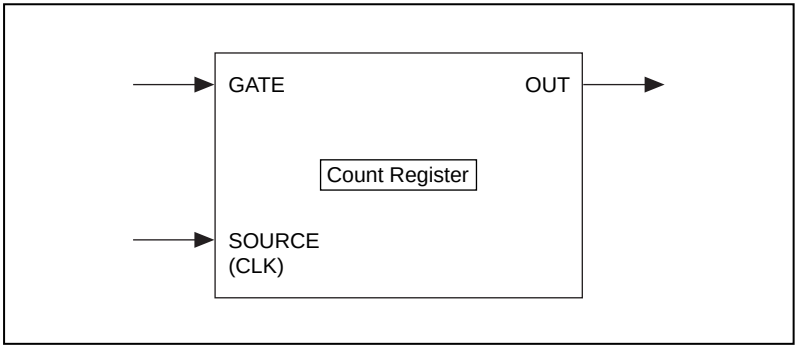



Although counters count only the signal transitions (edges) of a TTL source signal, you can use this counting capability in many ways:

- You can generate square TTL pulses for clock signals and triggers for other DAQ applications.
- You can measure the pulse width of TTL signals.
- You can measure the frequency and period of TTL signals.
- You can count TTL signal transitions (edges) or elapsed time.
- You can divide the frequency of TTL signals.

# Knowing the Parts of Your Counter

The following illustration shows a basic model of a counter.




A counter consists of a SOURCE (or CLK) input pin, a GATE input pin, an OUT output pin, and a count register. In plug-in device diagrams and in the *LabVIEW Function and VI Reference Manual*, these counter parts are called SOURCE*n* (or CLK*n*), GATE*n*, and OUT*n*, where *n* is the number of the counter.

Signal transitions (edges) are counted at the SOURCE input. The count register can be preloaded with a count value, and the counter increments or decrements the count register for each counted edge. The count register value always reflects the current count of signal edges. Reading the count register does not change its value. The GATE input can be used to control when counting occurs in your application. You can also use a counter with no gating, allowing the software to initiate the counting operation.

The OUT pin can be toggled according to available counter programming modes to generate various TTL pulses and pulse trains.

Use the OUT signal of a counter to generate various TTL pulse waveforms. If you are incrementing the count register value, you can configure the OUT signal to either toggle signal states or pulse when the count register reaches a certain value. The highest value of a counter is called the *terminal count* (TC). If you are decrementing, the count register TC value is 0. If you chose to have pulsed output, the counter outputs a high pulse that is equal in time to one cycle of the counter's SOURCE signal, which can be either an internal or external signal. If you chose a toggled output, the state of the output signal changes from high to low or low to high. For more control over the length of high and low outputs, use a toggled output. Refer to Chapter 24, *Generating a Square Pulse or Pulse Trains*, for more information.

Multiple counters can be concatenated for a greater counting range on most devices. For more information on how to concatenate counters, refer to Chapter 27, *Counting Signal Highs and Lows*.

# Knowing Your Counter Chip

Most National Instruments DAQ devices contain one of four different counter chips: the TIO-ASIC, the DAQ-STC, the Am9513, or the 8253/54 chip. Typically, 660*x* devices use the TIO-ASIC chip. E Series devices (for example the AT-MIO-16E-1) use the DAQ-STC chip. Legacy-type MIO devices (for example the AT-MIO-16) use the Am9513 chip. Low-cost Lab/1200-type devices (for example the Lab-PC-1200) use the 8253/54 chip. If you are not sure which chip your device uses, refer to your hardware documentation.

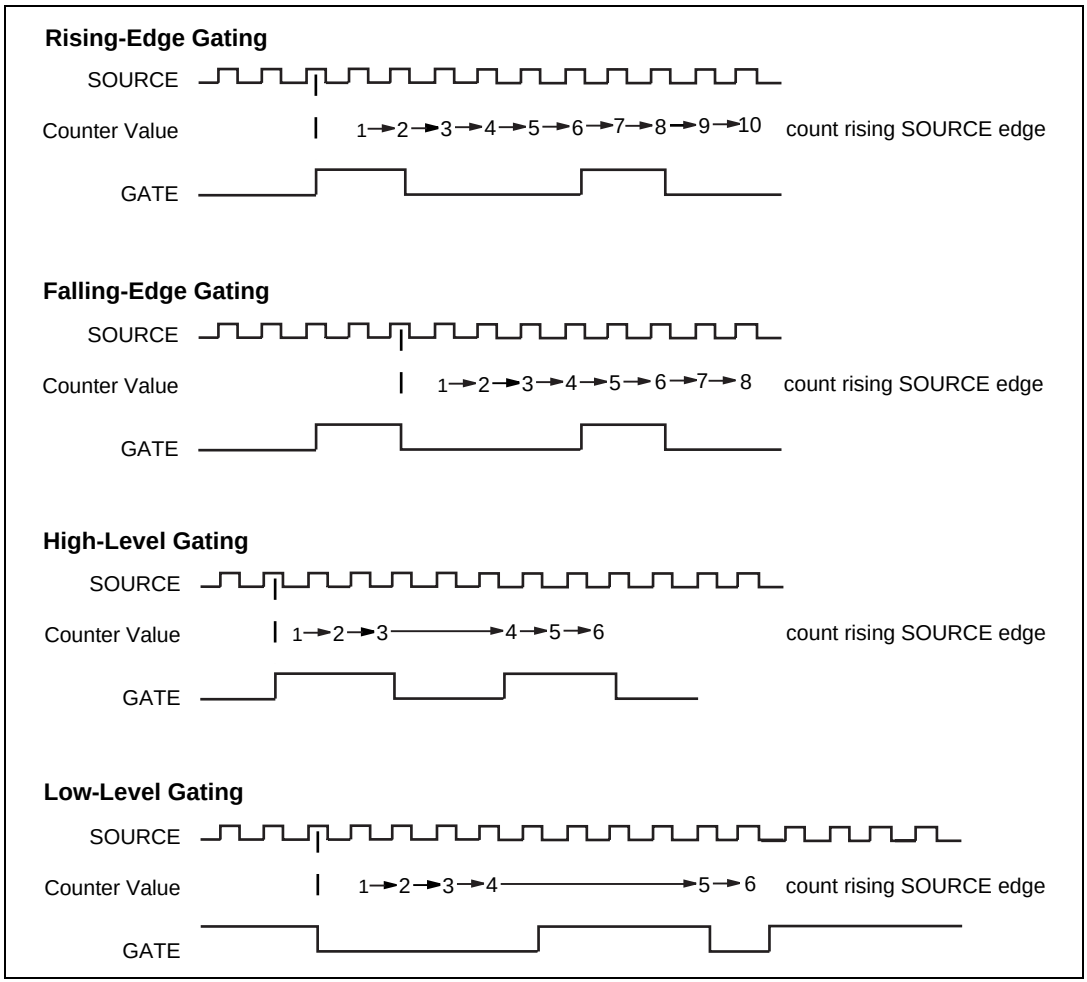



**Figure 23-1.** Counter Gating Modes

## TIO-ASIC

You can configure the TIO-ASIC to count either low-to-high or high-to-low transitions of the SOURCE input. The counter has a 32-bit count register with a counting range of 0 to $2^{32}$–1. It can be configured to increment or decrement for each counted edge. Furthermore, whether the count register increments or decrements can be controlled with an external digital line, which is useful for encoder applications. Of the gating modes shown in Figure 23-1, the gating modes the TIO-ASIC supports depends upon the application. You can set the configuration parameters described above using the Advanced VI, Counter Set Attribute.

## DAQ-STC

You can configure the DAQ-STC to count either low-to-high or high-to-low transitions of the SOURCE input. The counter has a 24-bit count register with a counting range of 0 to $2^{24}$–1. It can be configured to increment or decrement for each counted edge. Furthermore, whether the count register increments or decrements can be controlled with an external digital line, which is useful for encoder applications. Of the gating modes shown in Figure 23-1, the gating modes the DAQ-STC supports depends upon the application. You can set the configuration parameters discussed above using the Advanced VI, Counter Set Attribute.

## Am9513

You can configure the Am9513 to count either low-to-high or high-to-low transitions of the SOURCE input. The counter has a 16-bit count register with a counting range of 0 to 65535, and can be configured to increment or decrement for each counted edge. The Am9513 supports all of the gating modes shown in Figure 23-1. You can set the configuration parameters discussed above using the Advanced VI, CTR Mode Config.

## 8253/54

The 8253/54 chip counts low-to-high transitions of the CLK input. The counter has a 16-bit count register with a counting range of 65535 to 0 that decrements for each counted edge. Of the gating modes shown in Figure 23-1, the 8253/54 supports only high-level gating. For single pulse output, the 8253/54 can only create negative polarity pulses. For this reason, some applications require the use of a 7404 inverter chip to produce a positive pulse. The 14-pin 7404 is a common chip available from many electronics stores, and can be powered with the 5 V available on most DAQ devices. Figure 23-2 shows how to wire a 7404 chip to invert a signal.

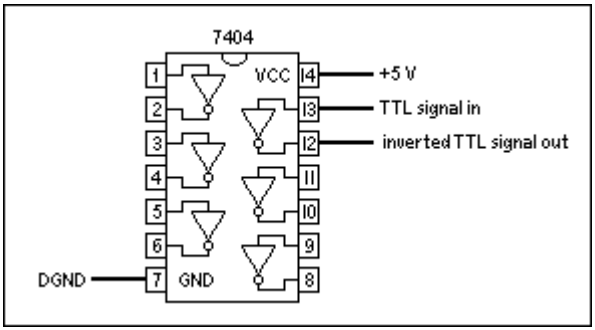



**Figure 23-2.** Wiring a 7404 Chip to Invert a TTL Signal

For specific information about the Counter VIs in LabVIEW, refer to Chapter 14, *Introduction to the LabVIEW Data Acquisition VIs*, in the *LabVIEW Function and VI Reference Manual*, or the LabVIEW *Online Reference*, available by selecting **Help»Online Reference**.

# 24

# Generating a Square Pulse or Pulse Trains

This chapter describes the ways you can generate a square pulse or multiple pulses (called *pulse trains*) using the counters available on your DAQ device with the example VIs in LabVIEW.

## Generating a Square Pulse

There are many applications where you may need to generate TTL pulses. TTL pulses can be used as clock signals, gates, and triggers. You can use a pulse train of known frequency to determine an unknown TTL pulse width. You also can use a single pulse of known duration to determine an unknown TTL signal frequency, or use a single pulse to trigger an analog acquisition.

There are two basic types of counter signal generation—*toggled* and *pulsed*. When a counter reaches a certain value, a counter configured for toggled output changes the state of the output signal, while a counter configured for pulsed output outputs a single pulse. The width of the pulse is equal to one cycle of the counter's SOURCE signal.

The following is a list of terms you should know before outputting a pulse or pulse train using LabVIEW:

- *phase 1* refers to the first phase or delay to the pulse.
- *phase 2* refers to the second phase or the pulse itself.
- *period* is the sum of phase 1 and phase 2.
- Frequency is the reciprocal of the period (1/period).
- In LabVIEW, you can adjust and control the times of phase 1 and phase 2 in your counting operation. You do this by specifying a *duty cycle*. The duty cycle equals

$$\frac{phase\ 2}{period}, \quad \text{where } period = phase\ 1 + phase\ 2$$

Examples of various duty cycles are shown in Figure 24-1. The first line shows a duty cycle of `0.5`, where, phase 1 and phase 2 are the same duration. A signal with a 0.5 duty cycle acts as a SOURCE for counter operations. The second line shows a duty cycle of `0.1`, where phase 1 has increased and phase 2 has decreased. The final line shows a large duty cycle of `0.9` where phase 1 is very short and the phase 2 duration is longer.

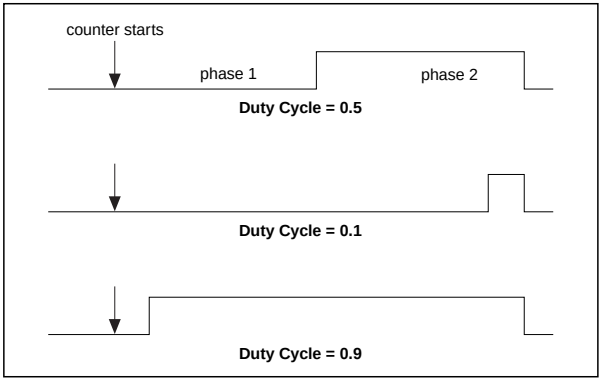



**Figure 24-1.** Pulse Duty Cycles

**Note** A high duty cycle denotes a long pulse phase relative to the delay phase.

How you generate a square pulse varies depending upon which counter chip your DAQ hardware has. If you are unsure which chip your device uses, refer to your hardware documentation.

## TIO-ASIC, DAQ-STC, and Am9513

When generating a pulse or pulse train with the TIO-ASIC, DAQ-STC, or Am9513 chip, you can define the polarity of the signal as positive or negative. Figure 24-2 shows these pulse polarities. Notice that for a signal with a positive polarity, the initial state is low, while a signal with negative polarity has a positive initial state.

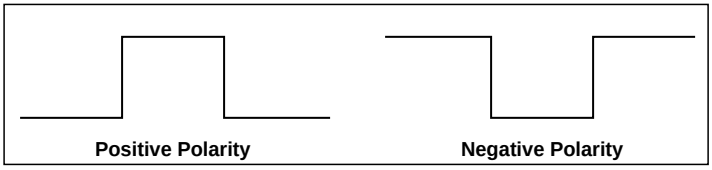



**Figure 24-2.** Positive and Negative Pulse Polarity

Each counter-generated pulse consists of two parts—phase 1 and phase 2. If the counter is configured to output a signal with positive polarity and toggled output, as shown in the following diagram, the period of time from when the counter starts counting to the first rising edge is called phase 1. The time between the rising and the following falling edge is called phase 2. If you configure the counter to generate a continuous pulse train, the counter repeats this process many times as shown on the bottom line of Figure 24-3.

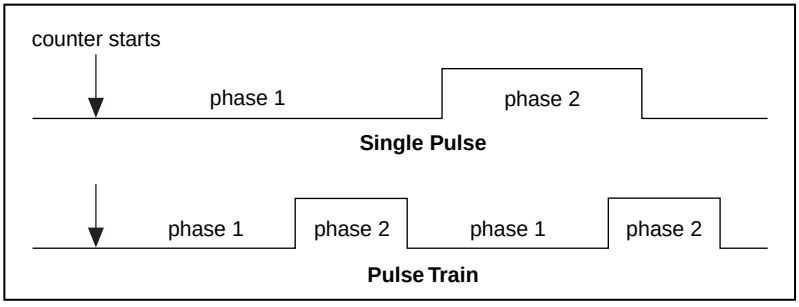



**Figure 24-3.** Pulses Created with Positive Polarity and Toggled Output

## 8253/54

When generating a pulse with the 8253/54 chip, the hardware limits you to a negative polarity pulse, as shown in Figure 24-2. The period of time from when the counter starts counting to the falling edge is called phase 1. The time between the falling and following rising edge is called phase 2. Figure 24-4 shows these phases for a single negative polarity pulse. If you need to create a positive polarity pulse, you can connect your negative polarity pulse to an external 7404 inverter chip.

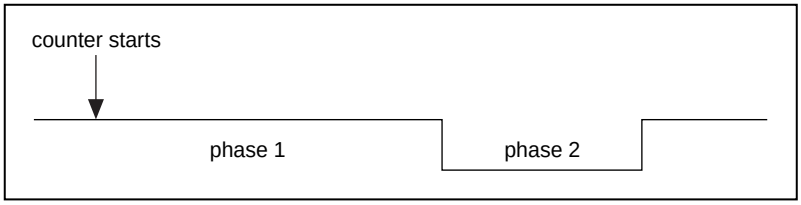



**Figure 24-4.** Phases of a Single Negative Polarity Pulse

When generating a pulse train with the 8253/54 chip, the hardware limits you to positive polarity pulses. Furthermore, the value loaded in the count register is divided equally to create phase 1 and phase 2. This means you will always get a 0.5 duty cycle if the count register is loaded with an even number. If you load the count register with an odd number, phase 1 will be longer than phase 2 by one cycle of the counter's CLK signal.

Now that you know the terms involving generating a single square pulse or a pulse train, you can learn about the LabVIEW VIs, and the physical connections needed to implement your application.

# Generating a Single Square Pulse

You can use a single pulse to trigger analog acquisition or to gate another counter operation. You also can use a single pulse to stimulate a device or circuit for which you need to acquire and test the response.

## TIO-ASIC, DAQ-STC, Am9513

Figure 24-5 shows two ways to connect your counter to generate a square pulse. In the Basic Connection, the edges of the internal SOURCE signal are counted to generate the output signal, the GATE is not used (software start), and the pulse signal on the OUT pin gets connected to your device. The Optional Connections use an external SOURCE from your device and is gated by your device. You can use either or both of these options.

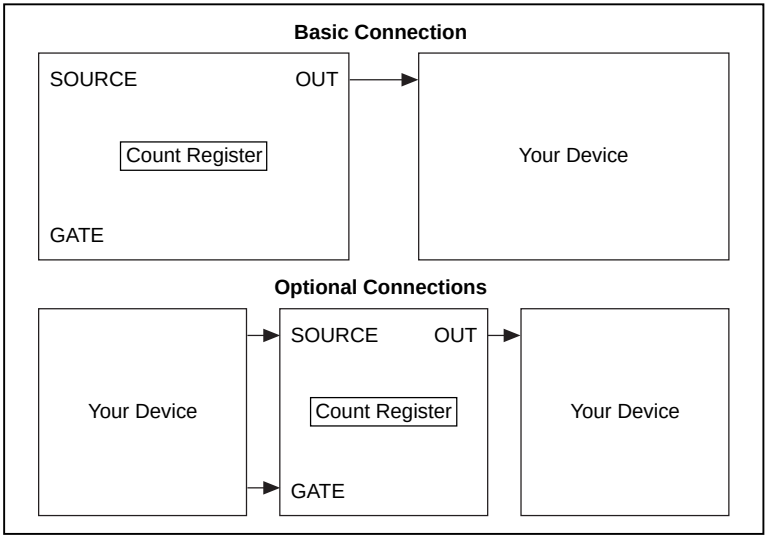



**Figure 24-5.** Physical Connections for Generating a Square Pulse

Figure 24-6 shows the block diagram of the Delayed Pulse-Easy (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb`. You also can use the example Delayed Pulse-Easy (9513) VI located in `labview\examples\daq\counter\Am9513.llb`. These examples use the Easy level Generate Delayed Pulse VI.

The Generate Delayed Pulse VI, found in **Functions»Data Acquisition»Counter**, tells your device to generate a single delayed pulse. This VI is self-contained and checks for errors automatically. With the Generate Delayed Pulse VI, you must connect the **pulse delay** (phase 1) and **pulse width** (phase 2) controls to define the output pulse. Sometimes the **actual pulse delay** and **pulse width** are not the same as you specified.

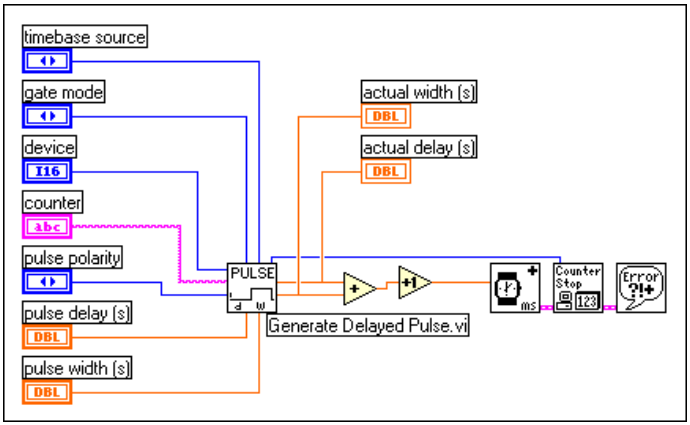



**Figure 24-6.** Diagram of Delayed Pulse-Easy (DAQ-STC) VI

To gain more control over when the counter begins generating a single square pulse, use Intermediate VIs instead of the Easy VIs. Figure 24-7 shows the diagram of the Delayed Pulse-Int (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb`. You can also use the example Delayed Pulse-Int (9513) VI located in `labview\examples\daq\counter\Am9513.llb`. These examples show how to generate a single pulse using Intermediate level VIs. The Delayed Pulse Generator Config VI configures the counter and the Counter Start VI generates the TTL signal. An example of this is generating a pulse after meeting certain conditions. If you used the Easy Counter VI, the VI configures and then immediately starts the pulse generation. With the Intermediate VIs, you can configure the counter long before the actual pulse generation begins. As soon as you want a pulse to be generated, the counter can immediately begin without having to configure the counter. In this situation, using Intermediate VIs improves performance.

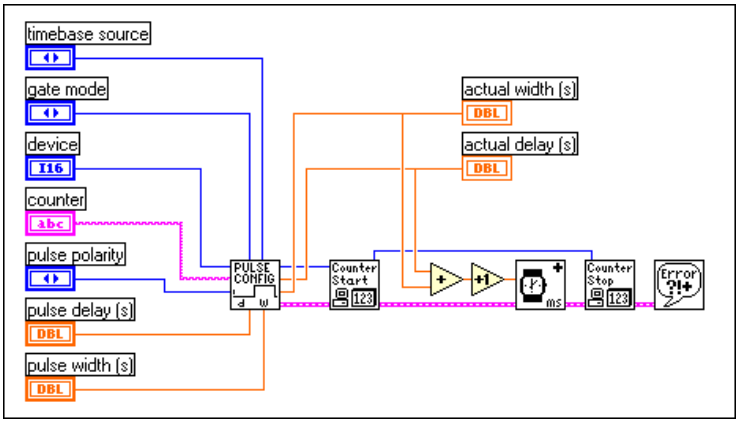



**Figure 24-7.** Diagram of Delayed Pulse-Int (DAQ-STC) VI

You must stop the counter if you want to use it for other purposes. For more information on stopping counters, refer to the *Stopping Counter Generations* section at the end of this chapter.

## 8253/54

The example Delayed Pulse (8253) VI located in `labview\examples\daq\counter\8253.llb` shows how to generate a negative polarity pulse. Due to the nature of the 8253/54 chip, three counters are used to generate this pulse. Because only **counter 0** is internally connected to a clock source, it is used to generate the timebase. **counter 1** is used to create the pulse delay that gates **counter 2**. **counter 2** is used to generate the pulse, which occurs on the OUT pin. Using multiple counters requires external wiring, which is shown in Figure 24-8 as well as being described on the front panel of the VI.

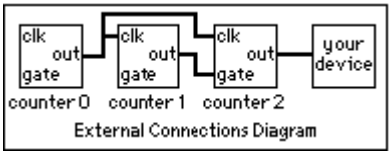



**Figure 24-8.** External Connections Diagram from the Front Panel of Delayed Pulse (8253) VI

This example uses a sequence structure to divide the basic tasks involved. Figure 24-9 shows frame 0 of the sequence where all of the counters are reset. Notice that counters 1 and 2 are reset so their output states start out high.

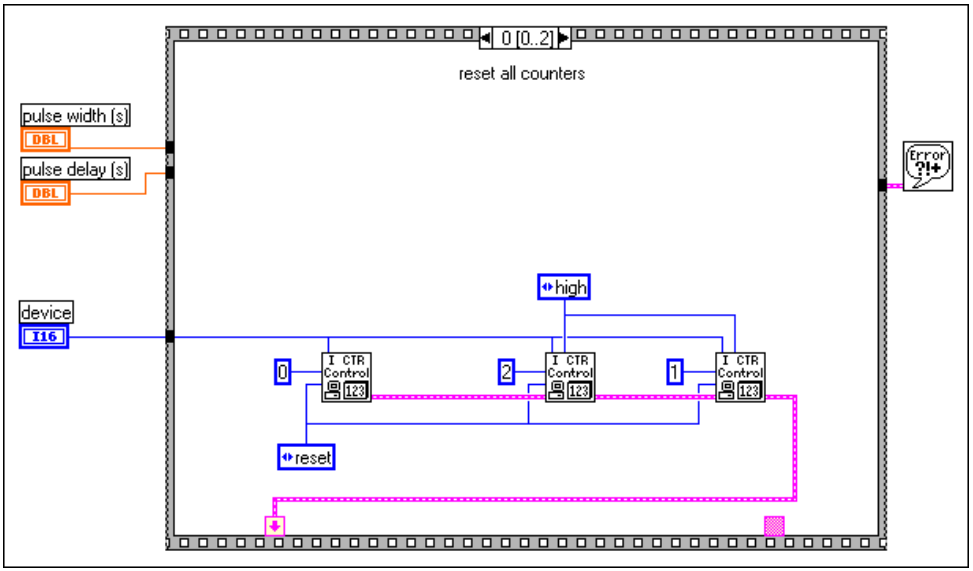



**Figure 24-9.** Frame 0 of Delayed Pulse (8253) VI

Figure 24-10 shows frame 1 of the sequence where the counters are set up for different counting modes. **counter 0** is set up to generate a timebase using the ICTR Timebase Generator subVI. **counter 1** is set up to toggle its output (low-to-high) when it reaches terminal count (TC). This toggled output is used to gate **counter 2**. **counter 2** is set up to output a low pulse when its gate goes high.

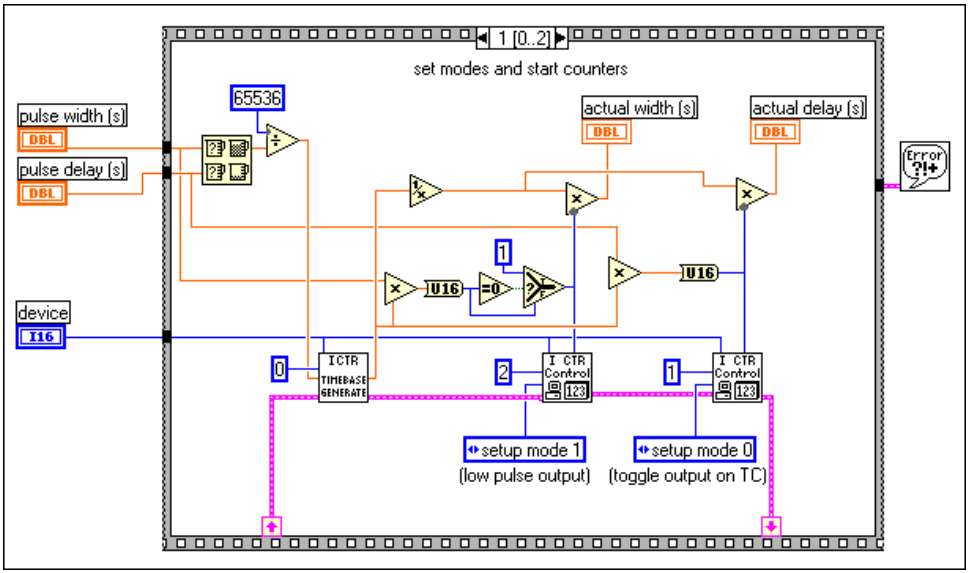



**Figure 24-10.** Frame 1 of Delayed Pulse (8253) VI

Figure 24-11 shows frame 2 of the sequence where a delay occurs so the delayed pulse has time to complete before the example can be run again. This is useful if the example is used as a subVI that is called repeatedly.

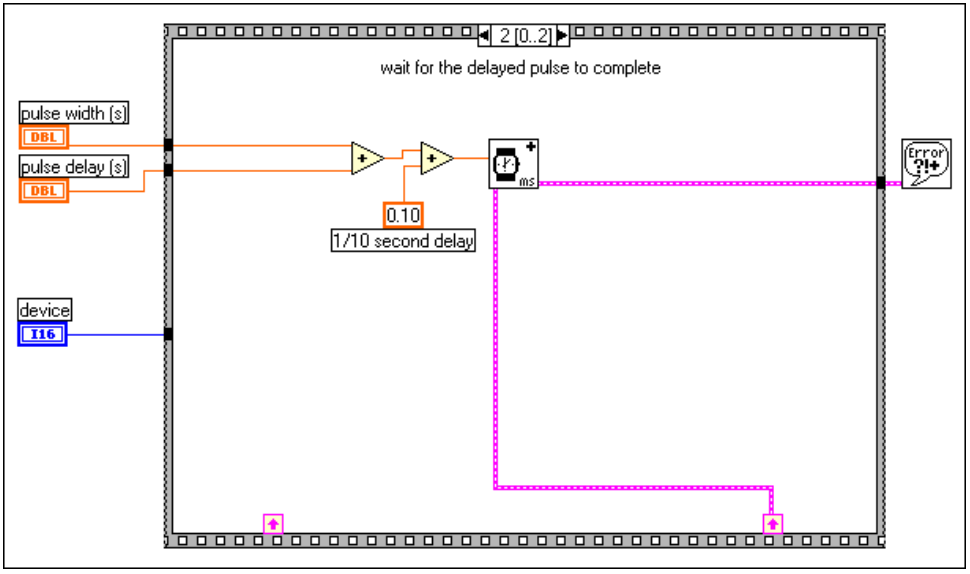



**Figure 24-11.** Frame 2 of Delayed Pulse (8253) VI

While this example works well for most pulses, it has limitations when your pulse delay gets very short (in the microsecond range), or when the ratio of pulse delay to pulse width gets very large. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

# Generating a Pulse Train

There are two types of pulse trains: *continuous* and *finite*. You can use a continuous pulse train as the SOURCE (CLK) of another counter or as the clock for analog acquisition (or generation). You can use a finite pulse train as the clock of an analog acquisition that acquires a predetermined number of points, or to provide a finite clock to an external circuit.

## Generating a Continuous Pulse Train

How you generate a continuous pulse varies depending upon which counter chip your DAQ hardware has. If you are not sure which chip your device uses, refer to your hardware manual.

## TIO-ASIC, DAQ-STC, Am9513

Figure 24-12 shows how to connect your counter and device to generate a continuous pulse train. The edges of the internal source signal are counted to generate the output signal. You obtain the continuous pulse train for your external device from the counter's OUT pin. You can optionally gate the operation with a signal connected to the GATE input pin. Instead of having an internal timebase as your SOURCE, you can connect an external signal.

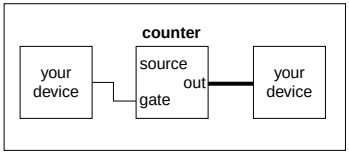



**Figure 24-12.** Physical Connections for Generating a Continuous Pulse Train

Figure 24-13 shows the diagram of the Cont Pulse Train-Easy (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb`. You can also use the example Cont Pulse Train-Easy (9513) VI located in `labview\examples\daq\counter\Am9513.llb`. These examples show how to use the Easy Counter VI, Generate Pulse Train, to specify the frequency, duty cycle, and pulse polarity of your pulse train. The **number of pulses** parameter defaults to 0 for continuous generation. When you click the **STOP** button, the While Loop stops, and a second call to Generate Pulse Train with the number of pulses set to –1 stops the counter.

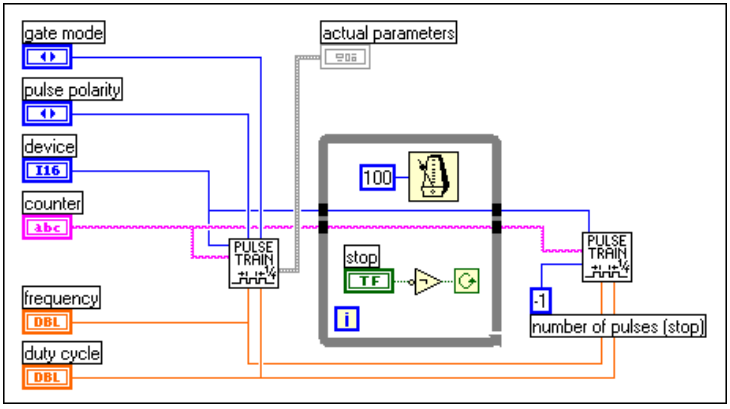



**Figure 24-13.** Diagram of Cont Pulse Train-Easy (DAQ-STC) VI

If you are generating a pulse train and want more control over when the counter starts, use the Intermediate VIs. Figure 24-14 shows the diagram of the Cont Pulse Train-Int (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb`. You could also use the example Cont Pulse Train-Int (9513) VI located in `labview\examples\daq\counter\Am9513.llb`. These examples show how to generate a simple pulse train using Intermediate VIs.

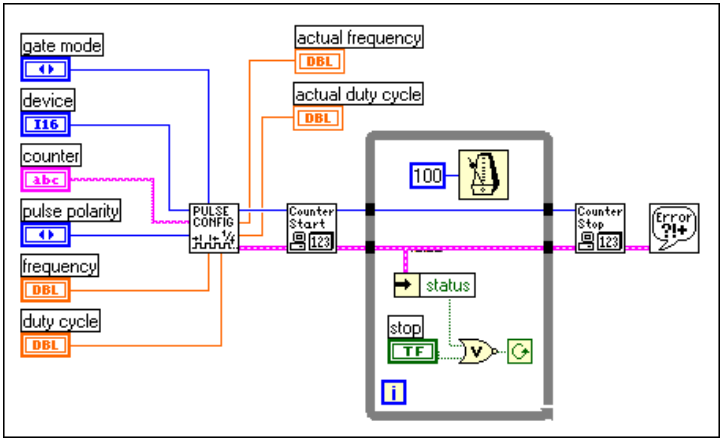



**Figure 24-14.** Diagram of Cont Pulse Train-Int (DAQ-STC) VI

With this VI you can specify the frequency, duty cycle, and pulse polarity of your pulse train. If the duty cycle is set to 0.0 or 1.0, the closest achievable duty cycle is used to generate a train of positive or negative pulses. The Continuous Pulse Generator Config VI configures the counter for the operation, and the Counter Start VI controls the initiation of the pulse train. For example, you may want to generate a continuous pulse train as the result of meeting certain conditions. If you use the Easy VI, the pulse train starts immediately. With the Intermediate VIs you can configure the counter at the beginning of your application, then wait to call Counter Start after the conditions are met. This approach will improve performance. When you click the **STOP** button, the While Loop stops, and Counter Stop is called to stop the pulse train.

You must stop the counter if you want to use it for other purposes. For more information on stopping counters, refer to the *Stopping Counter Generations* section at the end of this chapter.

## 8253/54

Figure 24-15 shows how to connect your counter and device to generate a continuous pulse train. If you use counter 0, an internal source is counted to generate the output signal. If you use counter 1 or 2, you will need to connect your own source to the CLK pin. You obtain the continuous pulse train for your external device from the counter's OUT pin.

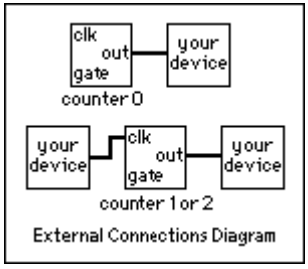



**Figure 24-15.** External Connections Diagram from the Front Panel of Cont Pulse Train (8253) VI

Figure 24-16 shows the diagram of the Cont Pulse Train (8253) VI located in `labview\examples\daq\counter\8253.llb`. This example shows how to use the Generate Pulse Train (8253) VI to generate a continuous pulse train. When using **counter 0** with this VI, you can specify the desired frequency. The actual frequency shows the closest frequency to your desired frequency that the counter was able to achieve. The actual duty cycle will be as close to 0.5 as possible for your actual frequency. When using **counter 1** or **counter 2**, you specify the divisor factor N to be used to divide your supplied source. You can optionally enter the user supplied timebase if you want the VI to calculate your actual frequency and actual duty cycle. When you click the **STOP** button, the While Loop stops, and a call to ICTR Control resets the counter, stopping the generation. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

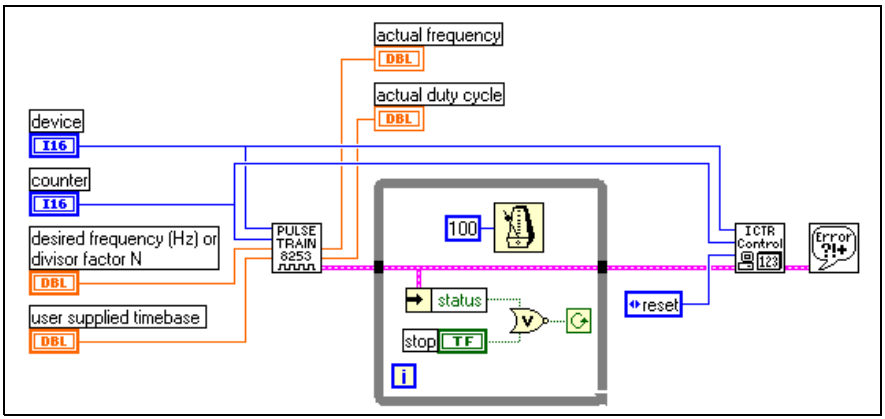



**Figure 24-16.** Diagram of Cont Pulse Train (8253) VI

## Generating a Finite Pulse Train

How you generate a finite pulse varies depending upon which counter chip your DAQ hardware has. If you are not sure which chip your device uses, refer to your hardware manual.

You can use the Easy I/O VI, Generate Pulse Train, or a stream of Intermediate VIs to generate a finite pulse train. With either technique, you must use two counters as shown in the connection diagram in Figure 24-17. Refer to Chapter 27, *Counting Signal Highs and Lows*, for more information on how to determine **counter-1** and how to use the adjacent counter VI. The maximum number of pulses in the pulse train is $2^{16}$–1, for Am9513 devices and $2^{24}$–1 for DAQ-STC devices.

Figure 24-17 shows the physical connections to produce a finite pulse train on the OUT pin of a counter. **counter** generates the finite pulse train with high-level gating. **counter-1** provides **counter** with a long enough gate pulse to output the number of desired pulses. You must externally connect the OUT pin of the **counter-1** to the GATE pin of **counter**. You also can gate **counter-1**.

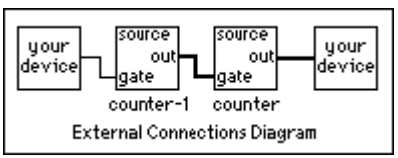



**Figure 24-17.** Physical Connections for Generating a Finite Pulse Train

## TIO-ASIC, DAQ-STC, Am9513

Figure 24-18 shows the diagram of the Finite Pulse Train-Easy (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb`. You can also use the example Finite Pulse Train-Easy (9513) VI located in `labview\examples\daq\counter\Am9513.llb`. These examples show how to use the Easy counter VI, Generate Pulse Train, to generate a finite pulse train. With this VI you can specify the number of pulses, frequency, duty cycle, and pulse polarity of your pulse train. The Wait+(ms) VI is used as a delay before the counters are reset. The Intermediate VI, Counter Stop, is called twice to stop the counters.

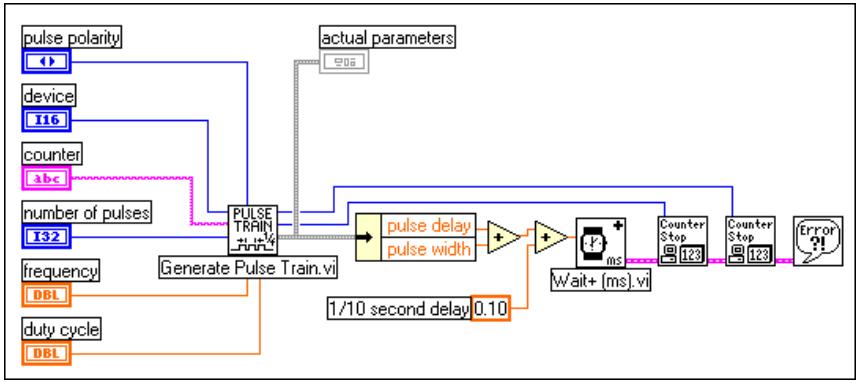



**Figure 24-18.** Diagram of Finite Pulse Train-Easy (DAQ-STC) VI

You can also create a finite pulse train using Intermediate VIs. Figure 24-19 shows the diagram of the Finite Pulse Train-Int (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb`. You could also use the example Finite Pulse Train-Int (9513) VI located in `labview\examples\daq\counter\Am9513.llb`. These examples show how to use the Intermediate VIs Continuous Pulse Generator Config and Delayed Pulse Generator Config.

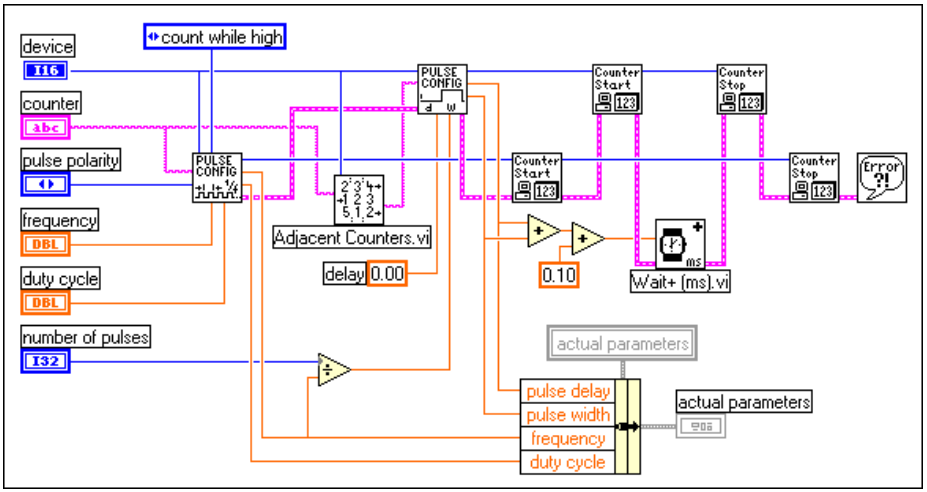



**Figure 24-19.** Diagram of Finite Pulse Train-Int (DAQ-STC) VI

In this operation, you use **counter** to generate a continuous pulse train with level gating while using **counter-1** to generate a minimum delayed pulse to gate the counter long enough to generate the desired number of pulses. The Continuous Pulse Generator Config VI configures **counter** to generate a continuous pulse train. Then, the Delayed Pulse Generator Config VI configures **counter-1** to generate a single delayed pulse. The first Counter Start VI in the flow begins the continuous pulse generation, and the next Counter Start VI generates a pulse after a specified time. The gate mode must be specified as level-gating on the Continuous Pulse Generator Config VI in order for the counter to wait for the gate signal from **counter-1**. The gate mode for the Delayed Pulse Generator Config VI can be set to a single or multiple edges. In other words, you could produce one finite pulse train or multiple pulse trains. The GATE signal for **counter-1** can be from an external device or from another counter on your DAQ device.

## DAQ-STC

With the DAQ-STC chip, you can internally route the OUT of one counter to the GATE of the next higher order counter, as shown in Figure 24-20. You can optionally GATE **counter-1**. While you use two counters, you do not need to externally wire between the OUT of **counter-1** and the GATE of **counter**.

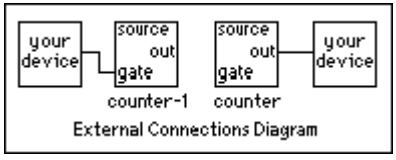



**Figure 24-20.** External Connections Diagram from the Front Panel of Finite Pulse Train Adv (DAQ-STC) VI

The example Finite Pulse Train-Adv (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb` takes advantage of internal wiring. Figure 24-21 shows the diagram of this example, which uses the Advanced counter VIs. The top row of **counter** VIs sets up **counter** to output a pulse train. Notice that the gate source input to the CTR Mode Config VI is set to `output of next lower order counter`. This sets the internal wiring such that **counter** will be gated by **counter-1**. The bottom row of counter VIs sets up **counter-1** to output a single pulse. The width of the pulse is calculated so it gates **counter** just long enough to output the chosen number of pulses. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

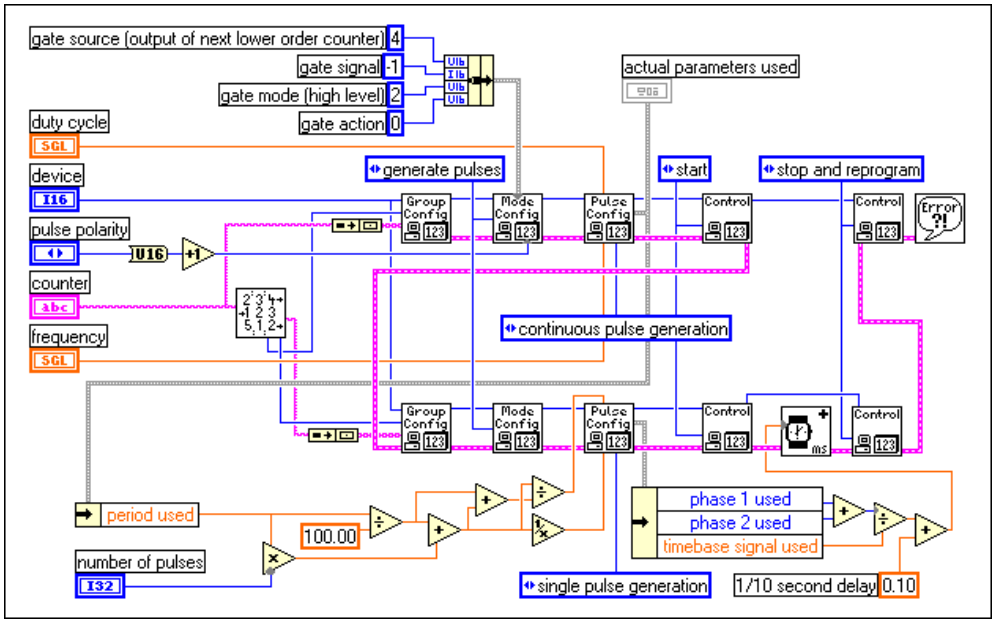



**Figure 24-21.** Diagram of Finite Pulse Train-Adv (DAQ-STC) VI

## 8253/54

Generating a finite pulse train with the 8253/54 chip uses all three counters. Figure 24-22 shows how to externally connect your counters. Because **counter 0** is internally connected to a clock source, **counter 0** is used to generate the timebase used by **counter 1** and **counter 2**. **counter 1** generates a single low pulse used to gate **counter 2**. Because **counter 2** must be gated with a high pulse, the output of **counter 1** is passed through a 7404 inverter chip prior to being connected to the GATE of **counter 2**. **counter 2** is set up to generate a pulse train at its OUT pin.

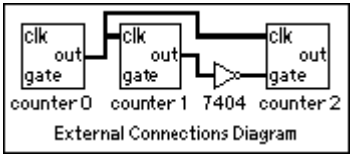



**Figure 24-22.** External Connections Diagram from the Front Panel of Finite Pulse Train (8253) VI

The example Finite Pulse Train (8253) VI located in `labview\examples\daq\counter\8253.llb` shows how to generate a finite pulse train. This example uses a sequence structure to divide the basic tasks involved. Figure 24-23 shows frame 0 of the sequence where all of the counters are reset. Notice **counter 1** is reset so its output state starts high.

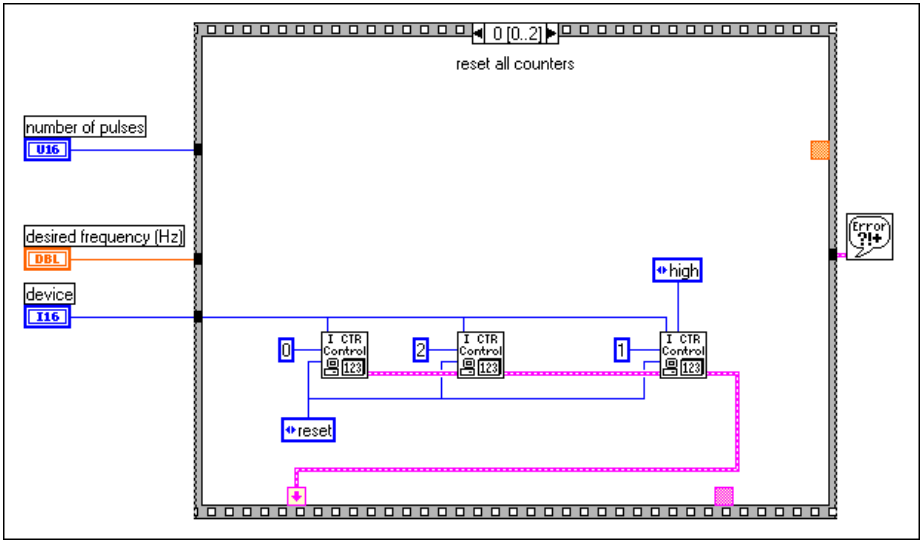



**Figure 24-23.** Frame 0 of Finite Pulse Train (8253) VI

Figure 24-24 shows frame 1 of the sequence where the counters are set up for different counting modes. **counter 0** is set up to generate a timebase using the ICTR Timebase Generator VI. **counter 1** is set up to output a single low pulse using the ICTR Control VI. **counter 2** is set up to output a pulse train using the ICTR Timebase Generator VI.

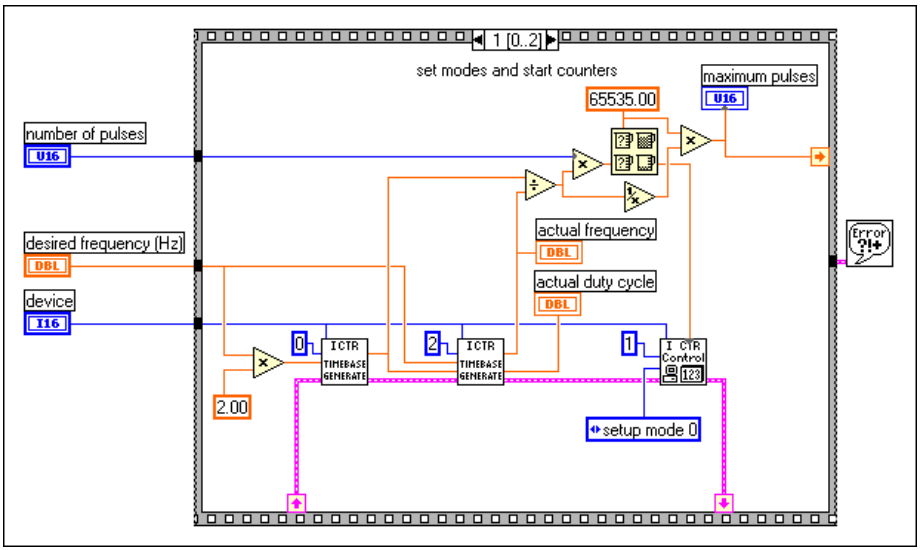



**Figure 24-24.** Frame 1 of Finite Pulse Train (8253) VI

Figure 24-25 shows frame 2 of the sequence where a delay occurs so that the finite pulse train has time to complete before the example can be run again. This is useful if the example is used as a subVI where it may get called over and over. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

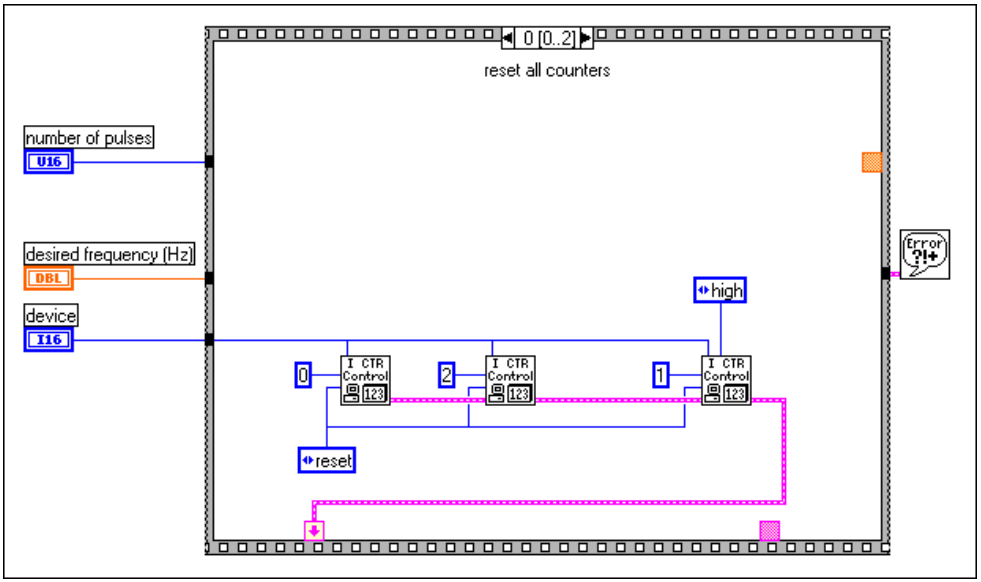



**Figure 24-25.** Frame 2 of Finite Pulse Train (8253) VI

# Counting Operations When All Your Counters Are Used

The DAQ-STC and Am9513 have counting operations available even when all the counters have been used.

TIO-ASIC and DAQ-STC devices feature a FREQ_OUT pin, and Am9513 devices feature an FOUT pin. You can generate a 0.5 duty cycle square wave on these pins without using any of the available counters.

The CTR Control VI, found in **Functions»Data Acquisition»Counters» Advanced Counters»9513 and Compatibility**, enables and disables the FOUT signal and sets the square wave frequency. The square wave frequency is defined by the FOUT timebase signal divided by the FOUT divisor. The front panel and block diagram below show an FOUT output configured to generate a 25,000 Hz square wave.

**Note** If you are using NI-DAQ 6.5 or higher, National Instruments recommends you use the new Advanced Counter VIs, such as Counter Group Config, Counter Get Attribute, Counter Set Attribute, Counter Buffer Read, and Counter Control. For more information, refer to the LabVIEW *Online Reference*.

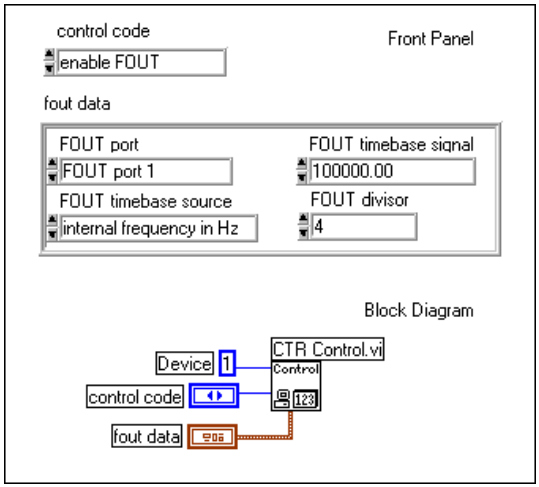



**Figure 24-26.** CTR Control VI Front Panel and Block Diagram

You can also refer to the Generate Pulse Train on FREQ_OUT VI located in `examples\daq\counter\DAQ-STC.llb`, or the Generate Pulse Train on FOUT VI located in `examples\daq\counter\Am9513.llb`. These examples generate a pulse train on these outputs.

# Knowing the Accuracy of Your Counters

When you generate a waveform, there can be an uncertainty of up to one timebase period between the start signal and the first counted edge of the timebase. This is due to the uncertainty in the exact relation of the start signal, which the software calls or the gate signal supplies to the first edge of the timebase, as shown in Figures 24-27.

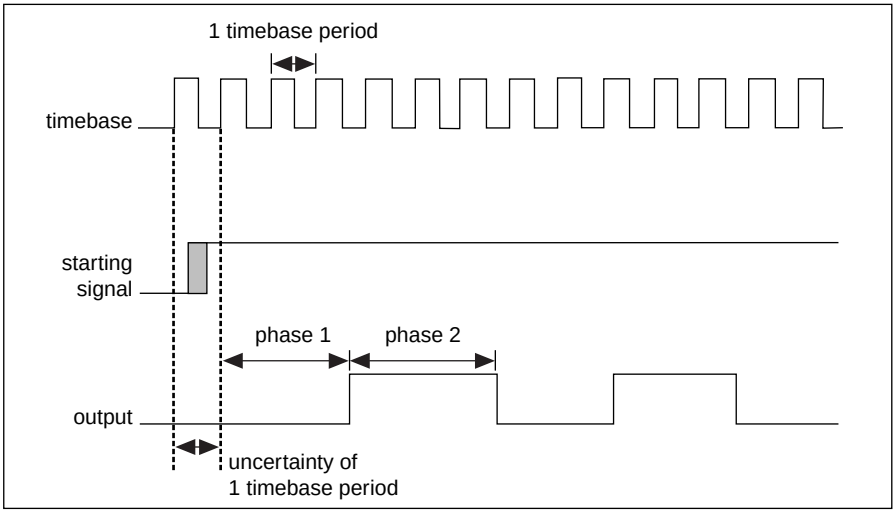



**Figure 24-27.** Uncertainty of One Timebase Period

## 8253/54

In addition to the previously described uncertainty, the 8253/54 chip has an additional uncertainty when used in mode 0. Mode 0 generates a low pulse for a chosen number of clock cycles, but a software delay is involved. This delay is because with mode 0 the counter output is set low by a software write to the mode setting. Afterward the count can be loaded and the counter starts counting down. The time between setting the output to low and loading the count is included in the output pulse. This time was found to be 20 μs when tested on a 200 MHz Pentium computer.

# Stopping Counter Generations

You can stop a counting operation in several ways. You can restart a counter for the same operation it just completed, you can reconfigure it to do something else, or you can call a specific VI to stop it. All of these methods allow you to use counters for different operations without resetting the entire device.

## DAQ-STC, Am9513

Figure 24-28 shows how to stop a counter using the Intermediate VI, Counter Stop. Notice that the Wait+ (ms) VI is called before Counter Stop. The Wait+ (ms) VI allows you to wire a time delay so that the previous counter operation has time to complete before the Counter Stop VI is called. The Wait+ (ms) and Counter Stop VIs are located in **Functions»Data Acquisition»Counter»Intermediate Counter**.

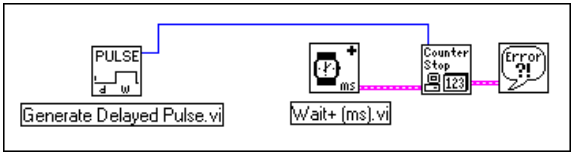



**Figure 24-28.** Using the Generate Delayed Pulse and Stopping the Counting Operation

To stop a generated pulse train, another Generate Pulse Train VI can be used with the number of pulses input set to –1, shown in Figure 24-29. This example expects that a pulse train is already being generated. The call to Generate Pulse Train VI stops the counter, and the call to Generate Delayed Pulse VI sets the counter up for a different operation.

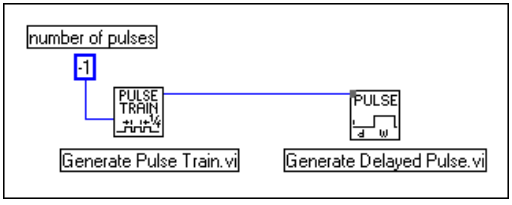



**Figure 24-29.** Stopping a Generated Pulse Train

## 8253/54

Calling ICTR Control VI with a control code of 7 (reset) can stop a counter on the 8253/54 chip. Examples are shown in Figures 24-9 and 24-23.

# 25

# Measuring Pulse Width

This chapter describes how you can use a counter to measure pulse width. There are several reasons you may need to determine pulse width. For example, to determine the duration of an event, set your application to measure the width of a pulse that occurs during that event. Another example is determining the interval between two events. In this case, you measure the pulse width between the two events. An example of when you might use this type of application is determining the time interval between two boxes on a conveyor belt or the time it takes one box to be processed through an operation. The event is an edge every time a box goes by a point, which prompts a digital signal to change in value.

## Measuring a Pulse Width

You can measure an unknown pulse width by counting the number of pulses of a faster known frequency that occur during the pulse to be measured. Connect the pulse you want to measure to the GATE input pin and a signal of known frequency to the SOURCE (CLK) input pin, as shown in Figure 25-1. The pulse of unknown width ($T_{pw}$) gates the counter configured to count a timebase clock of known period ($T_s$). The pulse width equals the timebase period times the count, or: $T_{pw} = T_s \times count$. The SOURCE (CLK) input can be an external or internal signal.

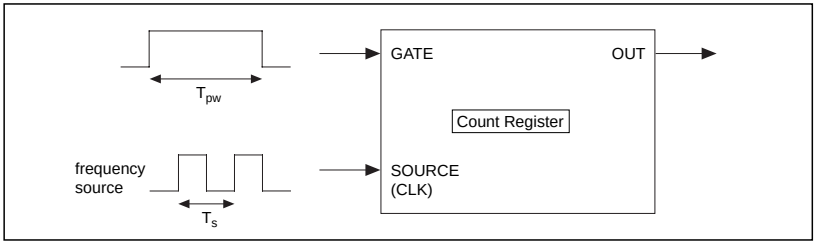



**Figure 25-1.** Counting Input Signals to Determine Pulse Width

An internal signal is based upon the type of counter chip on your DAQ device. With TIO-ASIC devices, you can choose internal timebases of 20 MHz, 100 kHz, and a device-specific maximum timebase. With DAQ-STC devices, you have a choice between internal timebases of

20 MHz and 100 kHz. With Am9513 devices, you can choose internal timebases of 1 MHz, 100 kHz, 10 kHz, 1 kHz, and 100 Hz. With 8253/54 devices, the internal timebase is either 2 MHz or 1 MHz, depending on which device you have.

Figure 25-2 shows how to physically connect the counter on your device to measure pulse width.

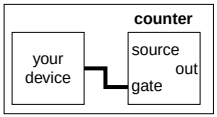



**Figure 25-2.** Physical Connections for Determining Pulse Width

# Determining Pulse Width

How you determine a pulse width depends upon which counter chip is on your DAQ device. If you are uncertain of which counter chip your DAQ device has, refer to your hardware documentation.

## TIO-ASIC, DAQ-STC

Figure 25-3 shows the diagram of the Measure Pulse-Easy (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb`, which uses the Easy VI, Measure Pulse Width or Period.

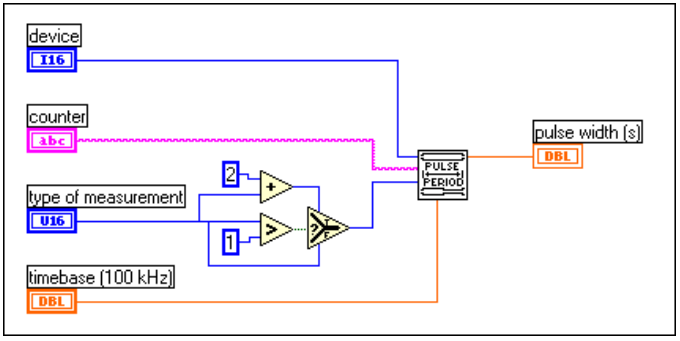



**Figure 25-3.** Diagram of Measure Pulse Width (DAQ-STC) VI

The Measure Pulse Width or Period VI counts the number of cycles of the specified timebase, depending on your choice from the **type of measurement** menu located on the front panel of the VI.

The **type of measurement** menu choices for this VI are shown in Figure 25-4.

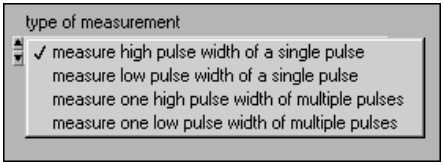

type of measurement
✓ measure high pulse width of a single pulse
measure low pulse width of a single pulse
measure one high pulse width of multiple pulses
measure one low pulse width of multiple pulses

**Figure 25-4.** Menu Choices for Type of Measurement for the Measure Pulse Width or Period (DAQ-STC) VI

Use the first two menu choices when you want to measure the width of a single pulse. In these cases, the GATE of the counter must start out in the opposite phase of the pulse you want to measure. For example, if you choose **measure high pulse width of a single pulse**, the GATE must start out low when you run the VI. If you attempt to measure a single high pulse, and the GATE is already high (such as in the middle of a pulse train) when you run the VI, an error will occur.

Use the last two menu choices when you want to measure the width of a single pulse within a train of multiple pulses. In these cases, it is the previous GATE transition that arms the counter to measure the next pulse. For example, if you choose **measure one high pulse width of multiple pulses**, the first high-to-low GATE transition from one pulse would arm the counter to measure the very next pulse.

The timebase you choose determines how long a pulse you can measure with the 24-bit counter. For example, the 100 kHz timebase allows you to measure a pulse up to $2^{24} \times 10\ \mu s = 167$ seconds long. The 20 MHz timebase allows you to measure a pulse up to 838 ms long. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

## Am9513

Figure 25-5 shows the diagram of the Measure Pulse-Easy (9513) VI located in `labview\examples\daq\counter\Am9513.llb`, which uses the Easy VI, Measure Pulse Width or Period.

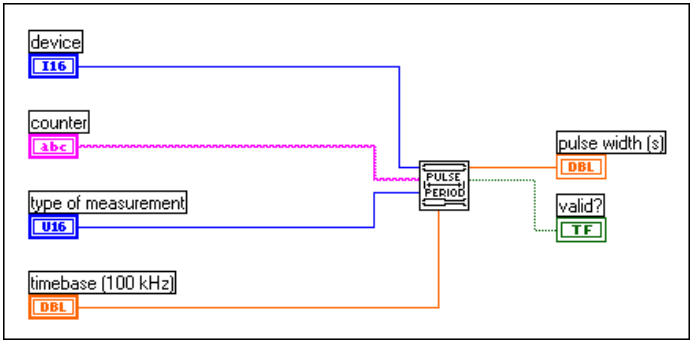



**Figure 25-5.** Diagram of Measure Pulse Width (9513) VI

The Measure Pulse Width or Period VI counts the number of cycles of the specified timebase, depending on your choice from the type of measurement menu located on the front panel of the VI. The type of measurement menu choices for this VI are shown in Figure 25-6.

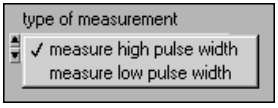



**Figure 25-6.** Menu Choices for Type of Measurement for the Measure Pulse Width or Period (9513) VI

Either menu choice can be used to measure the width of a single pulse, or to measure a pulse within a train of multiple pulses. However, the pulse must occur after the counter starts. Because the counter uses high-level gating, it might be difficult to measure a pulse within a fast pulse train. If the counter is started in the middle of a pulse, it measures the remaining width of that pulse.

The timebase you choose determines how long a pulse you can measure with the 16-bit counter. For example, the 100 Hz timebase allows you to measure a pulse up to $2^{16} \times 10\text{ms} = 655$ seconds long. The 1 MHz timebase allows you to measure a pulse up to 65 ms long. Because a faster timebase yields a more accurate pulse width measurement, it is best to use the fastest timebase possible without the counter reaching terminal count (TC).

The **valid?** output of the example VI indicates whether the counter measured the pulse without overflowing (reaching TC). However, **valid?** does not tell you whether a whole pulse was measured when measuring a pulse within a pulse train. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

## 8253/54

Figure 25-7 shows the diagram of the Measure Short Pulse Width (8253) VI located in `labview\examples\daq\counter\8253.llb`.

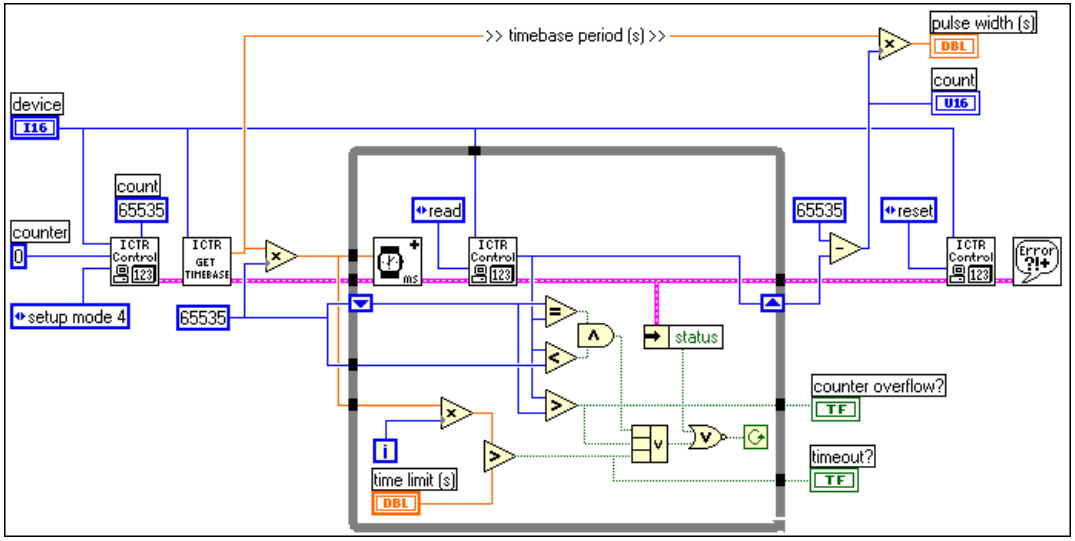



**Figure 25-7.** Diagram of Measure Short Pulse Width (8253) VI

This VI counts the number of cycles of the internal timebase of **Counter 0** to measure a high pulse width. You can measure a single pulse or a pulse within a train of multiple pulses. However, the pulse must occur after the counter starts. This means it may be difficult to measure a pulse within a fast pulse train because the counter uses high-level gating. To measure a low pulse width, insert a 7404 inverter chip between your pulse source and the GATE input of counter 0.

On the example diagram, the first call to ICTR Control VI sets up counting mode 4, which tells the counter to count down while the gate input is high. The Get Timebase (8253) VI is used to get the timebase of your DAQ device. A DAQ device with an 8253/54 counter has an internal timebase of either 1 MHz or 2 MHz, depending on the device. Inside the

While Loop, ICTR Control VI is called to continually read the count register until one of four conditions are met:

- The count register value has decreased but is no longer changing. It is finished measuring the pulse.
- The count register value is greater than the previously read value. An overflow has occurred.
- An error has occurred.
- Your chosen time limit has been reached.

After the While Loop, the final count is subtracted from the originally loaded count of 65535 and multiplied by the timebase period to yield the pulse width. Finally, the last ICTR Control VI resets the counter. Notice that this VI uses only **Counter 0**. If **Counter 0** has an internal timebase of 2 MHz, the maximum pulse width you can measure is $2^{16} \times 0.5\ \mu s = 32$ ms. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

# Controlling Your Pulse Width Measurement

How you control your pulse width measurement depends upon which counter chip is on your DAQ device. If you are uncertain of which counter chip you DAQ device has, refer to your hardware documentation.

## TIO-ASIC, DAQ-STC, or Am9513

Figure 25-8 shows one approach to measuring pulse width using the Intermediate VIs Pulse Width or Period Meas Config, Counter Start, Counter Read, and Counter Stop. You can use these VIs to control when the measurement of the pulse widths begins and ends. The Pulse Width or Period Config VI configures a counter to count the number of cycles of a known internal timebase. The Counter Start VI begins the measurement. The Counter Read VI determines if the measurement is complete and displays the count value. After the While Loop is stopped, the Counter Stop VI stops the counter operation. Finally, the General Error Handler VI notifies you of any errors.

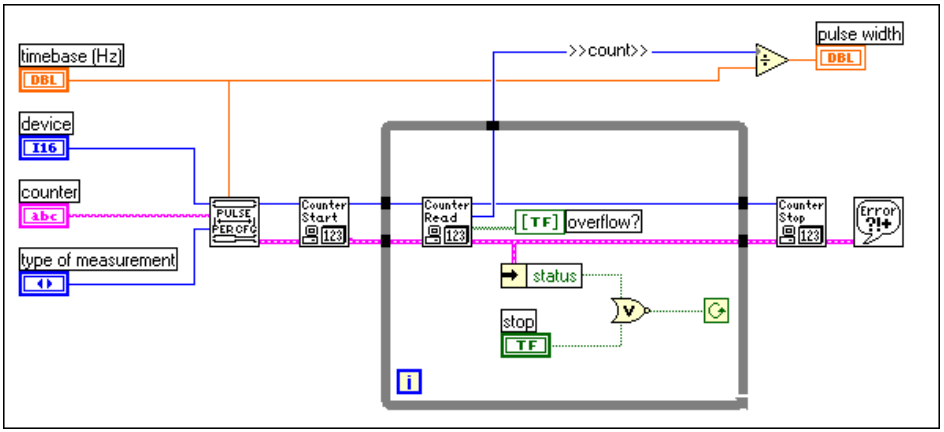



**Figure 25-8.** Measuring Pulse Width with Intermediate VIs

# Buffered Pulse and Period Measurement

With the TIO-ASIC and DAQ-STC chips, LabVIEW provides a buffer for counter operations. You would typically use buffered counter operations when you have a gate signal to trigger a counter several times. Figure 25-9 shows the diagram of the Meas Buffered Pulse-Period (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb`. You also can refer to examples located in `labview\examples\daq\counter\NI-TIO.llb`.

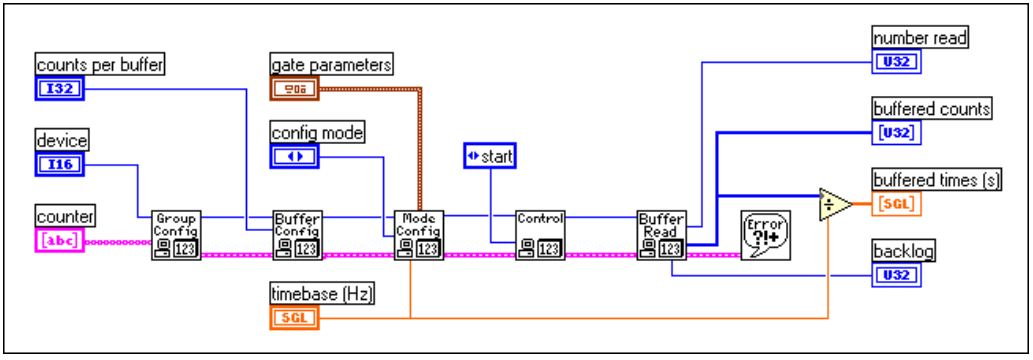



**Figure 25-9.** Diagram of Meas Buffered Pulse-Period (DAQ-STC) VI

With this example, you can perform four types of buffered measurements:

- **Buffered period measurement**—Measures a number of periods in a pulse train.
- **Buffered semi-period measurement**—Measures a number of high and low pulses in a pulse train.
- **Buffered pulse width measurement**—Measures a number of high or low pulses in a pulse train.
- **Buffered counting**—Each rising edge loads the current count into a finite buffer.

This example uses a single buffer. The block diagram uses the following Advanced VIs: CTR Group Config, CTR Buffer Config, CTR Mode Config, CTR Control, and CTR Buffer Read. CTR Group Config takes the counter and device and sets up a **taskID**. CTR Buffer Config sets up a finite buffer whose size is determined by the value you enter in **counts per buffer**. CTR Mode Config determines what type of counting operation to perform based on your choices for **gate parameters** and **config mode**. CTR Control starts the counting operation, but does not return until the counting has completed. CTR Buffer Read reads the buffer of data and returns the values to **buffered counts**. The **buffered times** are determined by dividing the counts by your chosen **timebase**. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

**Note** Continuous buffered operations are supported by the new Advanced Counter VIs in NI-DAQ 6.5 and higher.

**Note** If you are using NI-DAQ 6.5 or higher, National Instruments recommends you use the new Advanced Counter VIs, such as Counter Group Config, Counter Get Attribute, Counter Set Attribute, Counter Buffer Read, and Counter Control. For more information, refer to the LabVIEW *Online Reference*.

# Increasing Your Measurable Width Range

The maximum counting range of a counter and the chosen internal timebase determine how long of a pulse width can be measured. The internal timebase acts as the SOURCE. When measuring the pulse width of a signal, you count the number of source edges that occur during the pulse being measured. The counted number of SOURCE edges cannot exceed the counting range of the counter. Slower internal timebases allow you to measure longer pulse widths, but faster timebases give you a more accurate pulse width measurement. If you need a slower timebase than is available on your counter as shown in Table 25-1, set up an additional counter for pulse train generation and use the OUT of that counter as the SOURCE of the counter measuring pulse width.

**Table 25-1.** Internal Counter Timebases and Their Corresponding Maximum Pulse Width, Period, or Time Measurements

| Counter Type | Internal Timebases | Maximum Measurement |
|---|---|---|
| TIO-ASIC | Maximum* (depends on device) | — |
| | 20 MHz | 214.748 s |
| | 100 kHz | 11 h 55 m 49.67 s |
| DAQ-STC | 20 MHz | 838 ms |
| | 100 kHz | 167 s |
| Am9513 | 1 MHz | 65 ms |
| | 100 kHz | 655 ms |
| | 10 kHz | 6.5 s |
| | 1 kHz | 65 s |
| | 100 Hz | 655 s |
| 8253/54 | 2 MHz** | 32 ms |
| | 1 MHz** | 65 ms |

* You can obtain this timebase by calling the Counter Get Attribute VI.

** A DAQ device with an 8253/54 counter has one of these internal timebases available on counter 0, but not both.

# 26

# Measuring Frequency and Period

This chapter describes the various ways you can measure frequencies and periods of TTL signals using the counters on your DAQ device. One cycle of a signal, known as the period, is measured in units of time, usually seconds. The inverse of period is frequency, which is measured in cycles per second or hertz (Hz). The rate of your signal and the type of counter on your DAQ device determine whether you use frequency or period measurement. An example of when you would want to know the frequency of a signal is if you need to monitor the shaft speed of a motor.

## Knowing How and When to Measure Frequency and Period

A common way to measure the frequency of a signal is to measure the number of pulses that occur during a known time period. Figure 26-1 illustrates the measurement of a pulse train of an unknown frequency ($f_s$) by using a pulse of a known width ($T_G$). The frequency of the waveform equals the count divided by the known pulse width (frequency = count/$T_G$). The period is the reciprocal of the measured frequency (period = 1/$f_s$). You typically use frequency measurement for high frequency signals where the signal to be measured is approaching or faster than the chosen internal timebase.

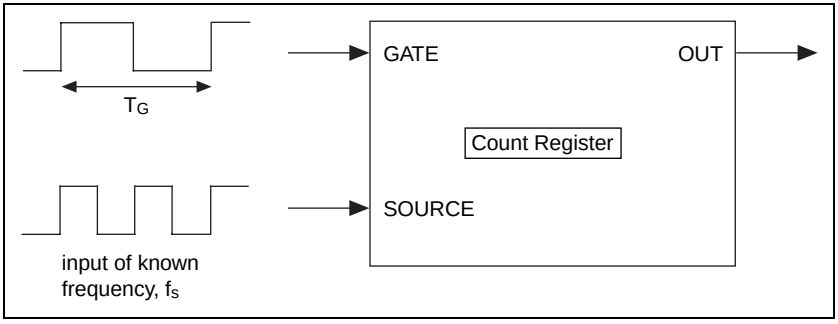



**Figure 26-1.** Measuring Square Wave Frequency

## TIO-ASIC, DAQ-STC, Am9513

For period measurement, you count the number of pulses of a known frequency ($f_s$) during one period of the signal to be measured. As shown in Figure 26-2, the signal of a known frequency is connected to the SOURCE, and the signal to be measured is connected to the GATE. The period is the count divided by the known frequency ($T_G$ = count/$f_s$).

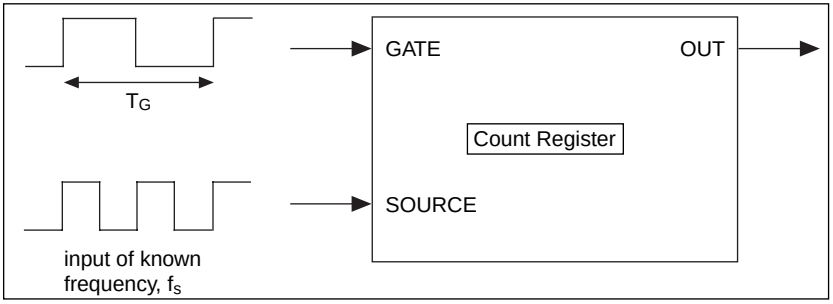



**Figure 26-2.** Measuring a Square Wave Period

You typically use period measurement for low frequency signals where the signal to be measured is significantly slower than the chosen internal timebase. The internal timebases for the TIO-ASIC are 20 MHz, 100 kHz, and a device-specific maximum timebase. The internal timebases for the DAQ-STC are 20 MHz and 100 kHz. The internal timebases for the Am9513 are 1 MHz, 100 kHz, 10 kHz, 1 kHz, and 100 Hz. Whether you use period measurement or frequency measurement, you can always obtain the other measurement by taking the inverse of the current one as shown in the following equations.

$$period\ measurement = \frac{1}{frequency\ measurement}$$

$$frequency\ measurement = \frac{1}{period\ measurement}$$

## 8253/54

The 8253/54 chip does not support period measurement, but you can use frequency measurement for a pulse train and take the inverse to get the period. The frequency examples discussed in this chapter calculate the period for you.

# Connecting Counters to Measure Frequency and Period

Figure 26-3 shows typical external connections for measuring frequency. In the figure, your device provides the signal with the frequency to be measured to the SOURCE (CLK) of **counter**. It can optionally control the GATE of **counter-1**. The OUT of **counter-1** supplies a known pulse to the GATE of **counter**. Finally, **counter** counts the number of cycles of the unknown pulse during the known GATE pulse.

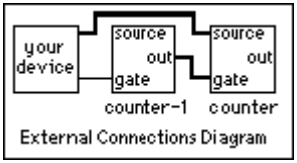



**Figure 26-3.** External Connections for Frequency Measurement

## TIO-ASIC, DAQ-STC, Am9513

Figure 26-4 shows typical external connections for measuring period. In the figure, your device provides the signal with the period to be measured to the GATE of **counter**. A timebase of known frequency is supplied to the SOURCE. This is usually an internal timebase, but it can be externally supplied. The counting range of your counter must not be exceeded during the period measurement. The range of the Am9513 is 65,335, the range of the DAQ-STC is 16,777,216, and the range of the TIO-ASIC is $2^{32}$–1. If the counting range is exceeded, select a slower internal timebase.

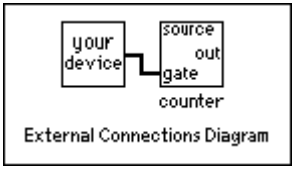



**Figure 26-4.** External Connections for Period Measurement

# Measuring the Frequency and Period of High Frequency Signals

How you measure the frequency and period of high frequency signals depends on the counter chip on your DAQ device. If you are unsure of which chip your DAQ device has, refer to your hardware documentation.

## TIO-ASIC, DAQ-STC

Figure 26-5 shows the Measure Frequency-Easy (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb`. This example uses the Easy VI, Measure Frequency, which can be found in **Functions» Data Acquisition»Counter**.

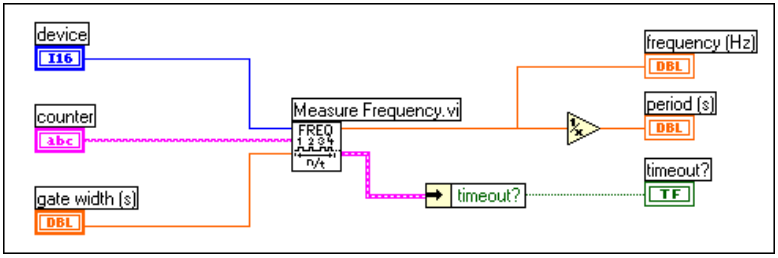



**Figure 26-5.** Diagram of Measure Frequency-Easy (DAQ-STC) VI

This VI initiates the counter to count the number of rising edges of a TTL signal at the SOURCE of counter during a known pulse at the GATE of **counter**. The width of that known pulse is determined by **gate width**. **Frequency** is the output for this example, and **period** is calculated by taking the inverse of the frequency. Remember, you must externally wire your signal to be measured to the SOURCE of **counter**, and the OUT of **counter-1** must be wired to the GATE of **counter**. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

## Am9513

Figure 26-6 shows the Measure Frequency-Easy (9513) VI located in `labview\examples\daq\counter\Am9513.llb`. This example uses the Easy VI, Measure Frequency, which can be found in **Functions» Data Acquisition»Counter**.

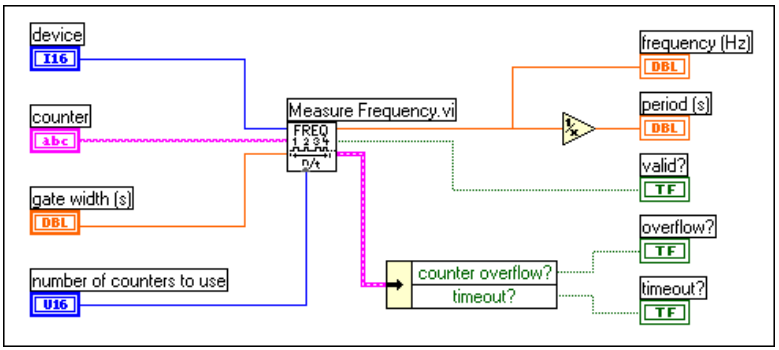



**Figure 26-6.** Diagram of Measure Frequency-Easy (9513) VI

This VI initiates the counter to count the number of rising edges of a TTL signal at the SOURCE of counter during a known pulse at the GATE of counter. The width of that known pulse is determined by gate width. Frequency is the output for this example, and period is calculated by taking the inverse of the frequency. The **valid?** output lets you know if the measurement completed without an overflow. The number of counters to use input lets you choose one counter for 16-bit measurement or two counters for 32-bit measurement. Remember that you must externally wire your signal to be measured to the SOURCE of counter, and the OUT of **counter-1** must be wired to the GATE of counter. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

## TIO-ASIC, DAQ-STC, Am9513

If you need more control over when your frequency measurement begins and ends, use the Intermediate VIs instead of the Easy VIs. Figure 26-7 shows one approach for this that uses the Event or Time Counter Config, Adjacent Counters, Delayed Pulse Generator Config, Counter Start, CTR Control, Counter Read, and Counter Stop VIs. The Delayed Pulse Generator Config VI configures **counter** to count the number of pulses while its GATE is high. The Adjacent Counters VI is used to determine the correct **counter-1**. The Delayed Pulse Generator Config VI then configures **counter-1** to generate a single pulse for the GATE signal. The Counter Start VI begins the counting operation for **counter** first, then **counter-1**. The CTR Control VI is an Advanced VI that is used to check if the GATE pulse has completed. The Counter Read VI returns the count value from **counter**, which is used to determine the frequency and pulse width. Finally, the Counter Stop VI stops the counter operation.

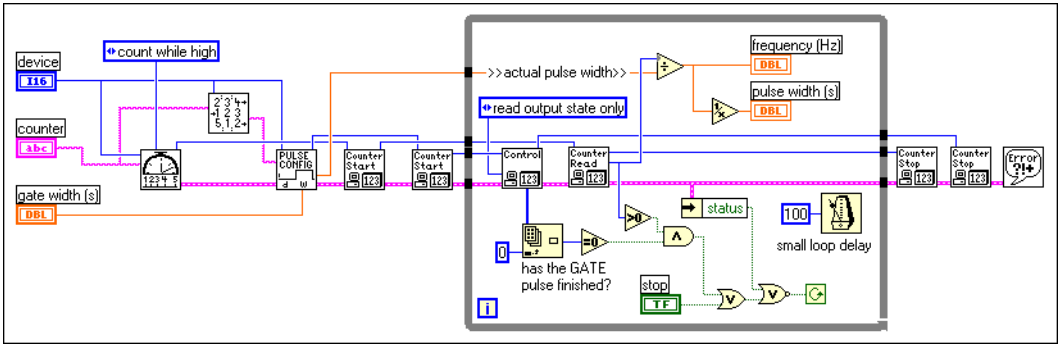



**Figure 26-7.** Frequency Measurement Example Using Intermediate VIs

## 8253/54

Figure 26-8 shows the Measure Frequency > 1 kHz (8253) VI located in `labview\examples\daq\counter\8253.llb`.

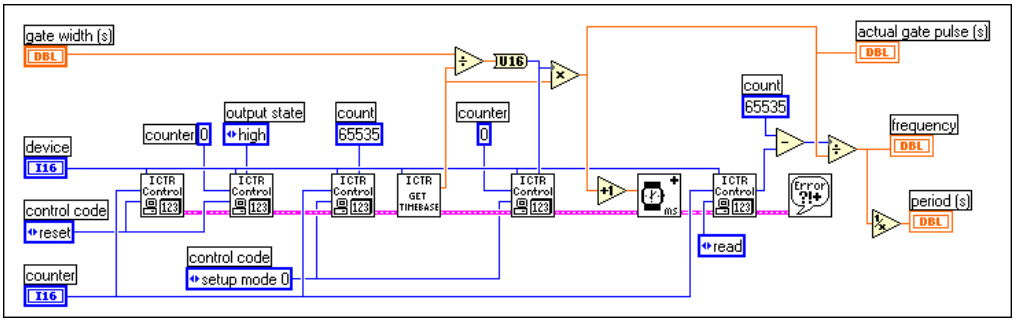



**Figure 26-8.** Diagram of Measure Frequency > 1 kHz (8253) VI

This VI initiates the counter to count the number of rising edges of a TTL signal at the CLK of **counter** during a known pulse at the GATE of **counter**. The known pulse is created by **counter 0**, and its width is determined by **gate width**. The maximum width of the pulse is 32 ms if your DAQ device has a 2 MHz internal timebase, and 65 ms if your DAQ device has a 1 MHz internal timebase. This maximum pulse is why this example only reads frequencies higher than 1 kHz. A frequency of 1 kHz generates 32 cycles during the 32 ms pulse. As this cycle count decreases (as with lower frequencies), the frequency measurement becomes less accurate. Frequency is the output for this example, and period is determined by taking the inverse of the frequency. You must externally wire the signal to be measured to the CLK of **counter**, and the OUT of **counter 0** must be wired through a 7404 inverter chip to the GATE of **counter**.

The diagram of the previous example uses the ICTR Control, Get Timebase (8253), and Wait + (ms) VIs. The first two ICTR Control VIs reset **counter** and **counter 0**. The next ICTR Control sets up **counter** to count down while its GATE input is high. The Get Timebase (8253) VI returns the internal timebase period for **counter 0** of device. This value is multiplied by the gate width to yield the count to be loaded into the count register of **counter 0**. The next ICTR Control VI loads this count and sets up **counter 0** to output a low pulse, during which cycles of the signal to be measured are counted.

One advantage of this example is that it only uses two counters. However, this example has two notable limitations. One limitation is that it cannot accurately measure low frequencies. If you need to measure lower frequencies, use the Measure Frequency < 1 kHz (8253) VI located in `labview\examples\daq\counter\8253.llb`. This VI uses three counters. The other limitation is that there is a software dependency, which causes **counter 0** to output a pulse slightly longer than the count it is given. This is the nature of the 8253 chip, and it can increase the readings of high frequencies. To avoid this software delay, use the Measure Frequency–Dig Start > 1 kHz (8253) located in `labview\examples\daq\counter\8253.llb`. For a complete description of each example, refer to the information found in **Windows»Show VI Info**.

# Measuring the Period and Frequency of Low Frequency Signals

How you measure the period and frequency of low frequency signals depends on which counter chip is on your DAQ device. If you are uncertain which chip your DAQ device has, refer to your hardware documentation.

## TIO-ASIC, DAQ-STC

Figure 26-9 shows the Measure Period-Easy (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb`. This example uses the Easy VI, Measure Pulse Width or Period, located in **Functions»Data Acquisition»Counter**.

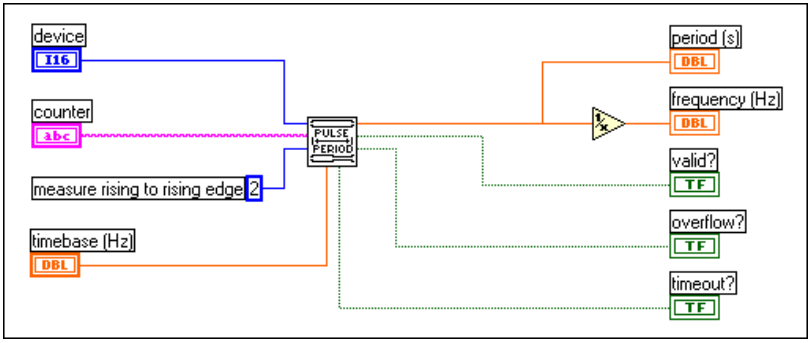



**Figure 26-9.** Diagram of Measure Period-Easy (DAQ-STC) VI

You connect your signal of unknown period to the GATE of **counter**. The counter measures the period between successive rising edges of your TTL signal by counting the number of internal **timebase** cycles that occur during the period. The **period** is the count divided by the timebase. The **frequency** is determined by taking the inverse of the **period**. You must choose **timebase** such that the counter does not reach its highest value, or terminal count (TC). Refer to Table 25-1, *Internal Counter Timebases and Their Corresponding Maximum Pulse Width, Period, or Time Measurements*, for maximum measurable periods for TIO-ASIC and DAQ-STC devices.

## Am9513

Figure 26-10 shows the example Measure Period-Easy (9513) VI located in `labview\examples\daq\counter\Am9513.llb`. This example uses the Easy VI, Measure Pulse Width or Period, located in **Functions»Data Acquisition»Counter**.

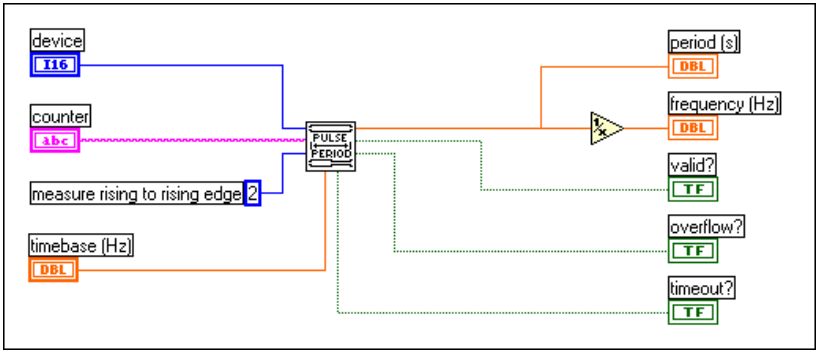



**Figure 26-10.** Diagram of Measure Period-Easy (9513) VI

You connect your signal of unknown period to the GATE of **counter**. The counter measures the period between successive rising edges of your TTL signal by counting the number of internal **timebase** cycles that occur during the period. The **period** is the count divided by the timebase. The **frequency** is determined by taking the inverse of the **period**. The **valid?** output indicates if the period was measured without overflow. Overflow occurs when the counter reaches its highest value, or terminal count (TC). You must choose **timebase** such that it does not reach TC. With a timebase of 1 MHz, the Am9513 can measure a period up to 65 ms. With a timebase of 100 Hz, you can measure a period up to 655 seconds.

## TIO-ASIC, DAQ-STC, Am9513

If you need more control over when period measurement begins and ends, use the Intermediate VIs instead of the Easy VIs. Figure 26-11 shows how to measure period and frequency.

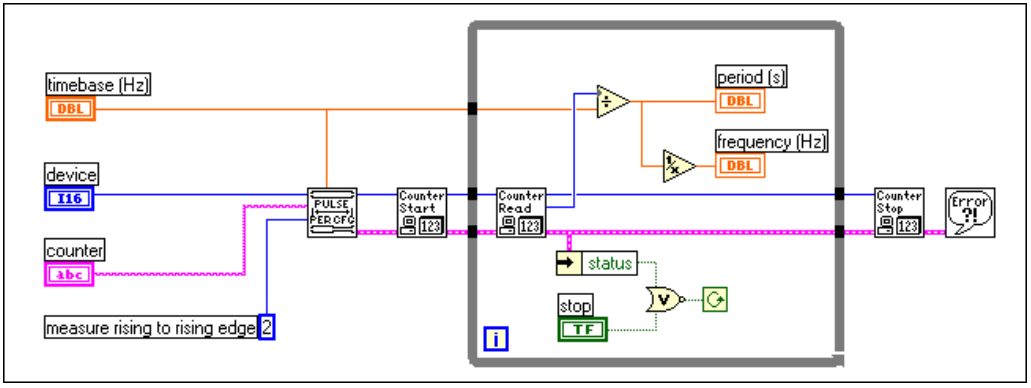



**Figure 26-11.** Measuring Period Using Intermediate Counter VIs

The Intermediate VIs used in Figure 26-11 include Pulse Width or Period Meas Config, Counter Start, Counter Read, and Counter Stop. The Pulse Width or Period Meas Config VI configures the counter for period measurement. The Counter Start VI begins the counting operation. The Counter Read VI returns the count value from the counter, which is used to determine the period and frequency. The Counter Stop VI stops the counter operation.

## 8253/54

The 8253/54 chip does not support period measurement, but you can use frequency measurement for a pulse train and take the inverse to get the period. The Measure Frequency < 1 kHz (8253) VI located in `labview\examples\daq\counter\8253.llb` measures frequency and calculates the period for you.

# 27

# Counting Signal Highs and Lows

This chapter describes the various ways you can count TTL signals using the counters on your DAQ device. Counters can count external events such as rising and falling edges on the SOURCE (CLK) input pin. They can also count elapsed time using the rising and falling edges of an internal timebase. An example of counting events is calculating the output of a production line. An example of counting time is calculating how long it takes to produce one item on a production line.

## Connecting Counters to Count Events and Time

Figure 27-1 shows typical external connections for counting events. In the figure, **your device** provides the TTL signal to be counted, and it is connected to the SOURCE (CLK) of **counter**. The number of events counted is determined by reading the count register of **counter**.

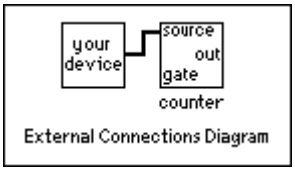



**Figure 27-1.** External Connections for Counting Events

Figure 27-2 shows typical external connections for counting elapsed time. In the figure, your device provides a pulse to the GATE of **counter**. While the gate pulse is high, **counter** counts a known internal timebase. Dividing the count by the internal timebase determines the elapsed time.

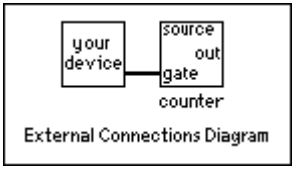



**Figure 27-2.** External Connections for Counting Elapsed Time

## Am9513

With the Am9513, you can extend the counting range of a counter by connecting the OUT of one counter to the SOURCE of the next higher order counter (counter+1). This is called *cascading* counters. By cascading counters you can increase your counting range from a 16-bit counting range of 65,535 to a 32-bit counting range of 4,294,967,295. The Am9513 chip has a set of five counters where higher order counters can be cascaded. The TIO-10 device has two Am9513 chips for a total of 10 counters. Table 27-1 identifies adjacent counters on the Am9513 (one and two chips). This information is useful when cascading counters.

**Table 27-1.** Adjacent Counters for Counter Chips

| Next Lower Counter | Counter | Next Higher Counter |
|---|---|---|
| 5 | 1 | 2 |
| 1 | 2 | 3 |
| 2 | 3 | 4 |
| 3 | 4 | 5 |
| 4 | 5 | 1 |
| 10 | 6 | 7 |
| 6 | 7 | 8 |
| 7 | 8 | 9 |
| 8 | 9 | 10 |
| 9 | 10 | 6 |

Figure 27-3 shows typical external connections for cascading counters when counting events. Notice that the OUT of **counter** is connected to the SOURCE of **counter+1**.

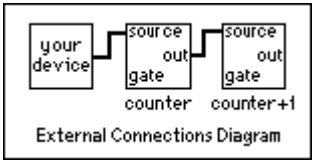



**Figure 27-3.** External Connections to Cascade Counters for Counting Events

Figure 27-4 shows typical external connections for cascading counters when counting elapsed time. Notice that the OUT of **counter** is connected to the SOURCE of **counter+1**.

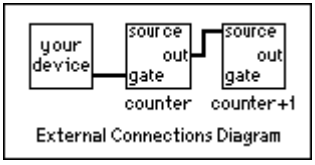



**Figure 27-4.** External Connections to Cascade Counters for Counting Elapsed Time

# Counting Events

How you count events depends upon which counter chip is on your DAQ device. If you are uncertain which counter your DAQ device has, refer to your hardware documentation.

## TIO-ASIC, DAQ-STC

Figure 27-5 shows the Count Events-Easy (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb`. This example uses the Easy VI, Count Events or Time, which can be found in **Functions»Data Acquisition»Counter**.

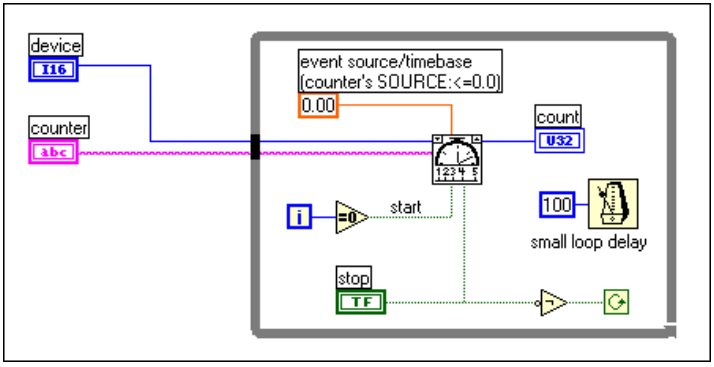



**Figure 27-5.** Diagram of Count Events-Easy (DAQ-STC) VI

This VI initiates the counter to count the number of rising edges of a TTL signal at the SOURCE of **counter**. The counter continues counting until you click the **STOP** button. You must externally wire your signal to be counted to the SOURCE of **counter**. For a description of this example, refer to the information found in **Windows»Show VI Info**.

If you need more control over when your event counting begins and ends, use the Intermediate VIs instead of the Easy VIs. Figure 27-6 shows the Count Events-Int (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb`.

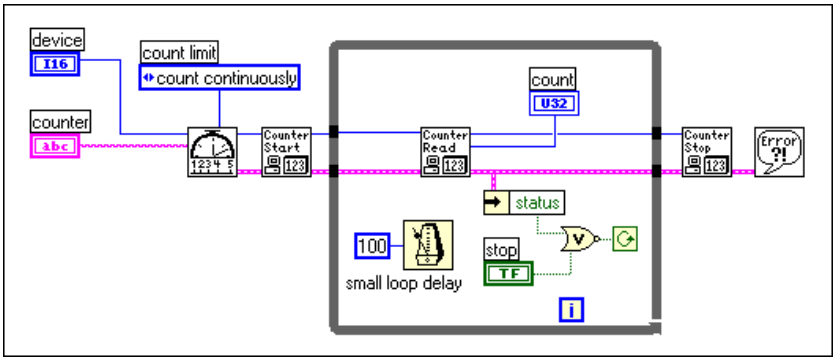



**Figure 27-6.** Diagram of Count Events-Int (DAQ-STC) VI

This example uses the Intermediate VIs Event or Time Counter Config, Counter Start, Counter Read, and Counter Stop. The Event or Time Counter Config VI configures **counter** to count the number of rising edges of a TTL signal at its SOURCE input pin. The Counter Start VI begins the counting operation for **counter**. The Counter Read VI returns the count until you click the **STOP** button or an error occurs. Finally, the Counter Stop VI stops the counter operation. Remember that you must externally wire your signal to be counted to the SOURCE of **counter**. You can optionally gate **counter** with a pulse to control when it starts and stops counting. To do this, wire your pulse to the GATE of counter, and choose the appropriate **gate mode** from the front panel menu. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

## Am9513

Figure 27-7 shows the Count Events-Easy (9513) VI located in `labview\examples\daq\counter\Am9513.llb`. This example uses the Easy VI, Count Events or Time, which can be found in **Functions» Data Acquisition»Counter**.

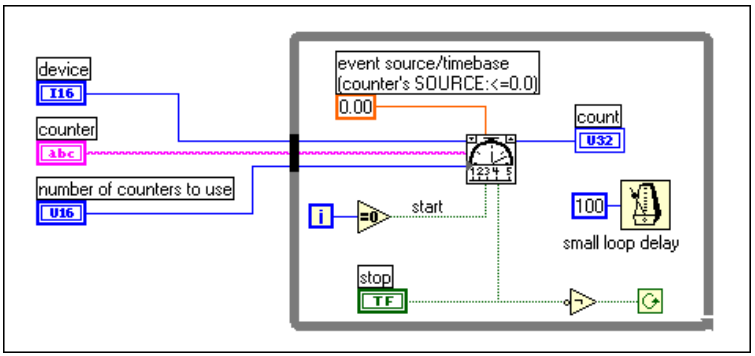



**Figure 27-7.** Diagram of Count Events-Easy (9513) VI

This VI initiates the counter to count the number of rising edges of a TTL signal at the SOURCE of counter. The counter continues counting until you click the **STOP** button. You must externally wire your signal to be counted to the SOURCE of **counter**. Additionally, you can cascade two counters by choosing **two counters (32-bits)** in the **number of counters to use** menu. This will extend your counting range to over 4 billion. You must also wire the OUT of **counter** to the SOURCE of **counter+1** for this increased counting range. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

If you need more control over when your event counting begins and ends, use the Intermediate VIs instead of the Easy VIs. Figure 27-8 shows the Count Events-Int (9513) VI located in `labview\examples\daq\counter\Am9513.llb`.

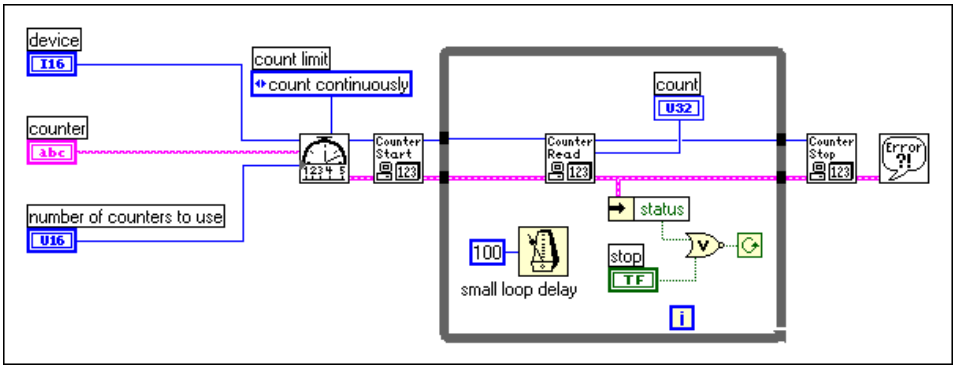



**Figure 27-8.** Diagram of Count Events-Int (9513) VI

This example uses the Intermediate VIs Event or Time Counter Config, Counter Start, Counter Read, and Counter Stop. The Event or Time Counter Config VI configures **counter** to count the number of rising edges of a TTL signal at its SOURCE input pin. The Counter Start VI begins the counting operation for **counter**. The Counter Read VI returns the count until you click the **STOP** button or an error occurs. Finally, the Counter Stop VI stops the counter operation. You must externally wire your signal to be counted to the SOURCE of **counter**. You can optionally gate counter with a pulse to control when it starts and stops counting. To do this, wire your pulse to the GATE of **counter**, and choose the appropriate **gate mode** from the front panel menu. Additionally, you can cascade two counters by choosing **two counters (32-bits)** in the **number of counters to use** menu. This will extend your counting range to over 4 billion. You must also wire the OUT of **counter** to the SOURCE of **counter+1** for this increased counting range. For a complete description of this example, read the information found in **Window»Show VI Info**.

## 8253/54

Figure 27-9 shows the Count Events (8253) VI located in `labview\examples\daq\counter\8253.llb`. This example uses the Intermediate VI, ICTR Control, found in **Functions» Data Acquisition»Counter»Intermediate Counter**.

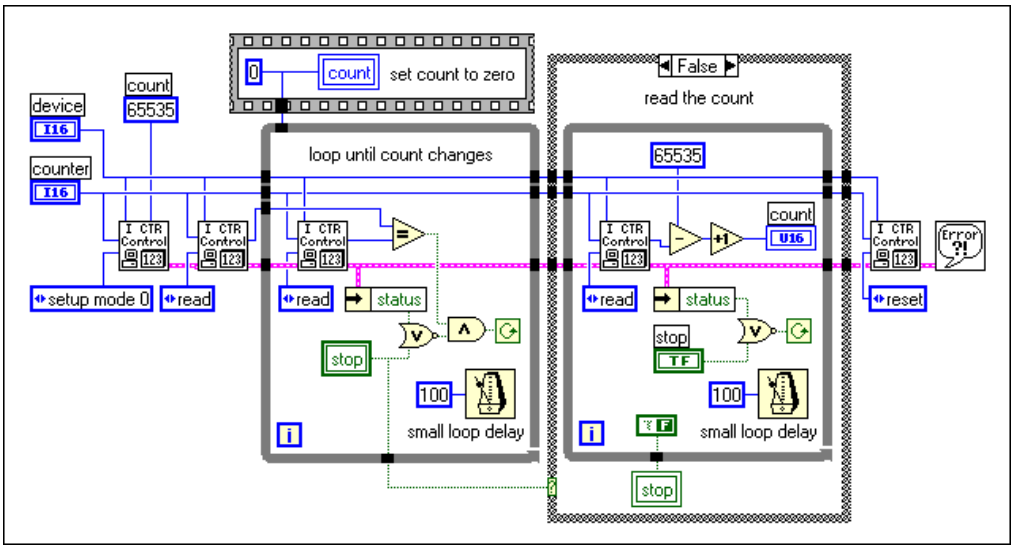



**Figure 27-9.** Diagram of Count Events (8253) VI

This VI initiates the counter to count the number of rising edges of a TTL signal at the CLK of **counter**. Looking at the diagram, the first call to ICTR Control loads the count register and sets up **counter** to count down. The second call to ICTR Control reads the count register. Inside the first While Loop, the count is read until it changes. While the count register has been previously loaded, the new value is not active until the first edge is counted on the CLK pin. Once the first edge comes in, the second While Loop takes over and continually reads the count until you click the **STOP** button or an error occurs. You must externally wire your signal to be counted to the CLK of counter. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

# Counting Elapsed Time

How you count elapsed time depends upon which counter chip is on your DAQ device. If you are unsure of which chip your DAQ device has, refer to your hardware documentation.

## TIO-ASIC, DAQ-STC

Figure 27-10 shows the Count Time-Easy (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb`. This example uses the Easy VI, Count Events or Time, which can be found in **Functions» Data Acquisition»Counter**.

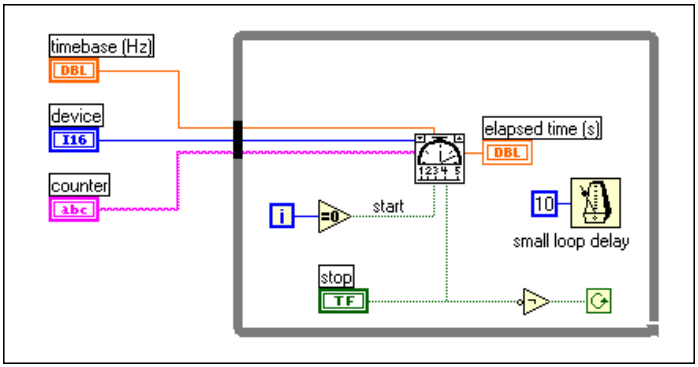



**Figure 27-10.** Diagram of Count Time-Easy (DAQ-STC) VI

This VI initiates the counter to count the number of rising edges of a known internal timebase at the SOURCE of **counter**. The Count Events or Time VI takes care of dividing the count by the timebase frequency to determine the **elapsed time**. The counter continues timing until you click the **STOP** button. You do not need to make any external connections. The length of time that can be counted depends on the maximum count of the counter and the chosen **timebase**. Refer to Table 25-1, *Internal Counter Timebases and Their Corresponding Maximum Pulse Width, Period, or Time Measurements*, for maximum measurable time information.

If you need more control over when your elapsed timing begins and ends, use the Intermediate VIs instead of the Easy VIs. Figure 27-11 shows the Count Time-Int (DAQ-STC) VI located in `labview\examples\daq\counter\DAQ-STC.llb`.

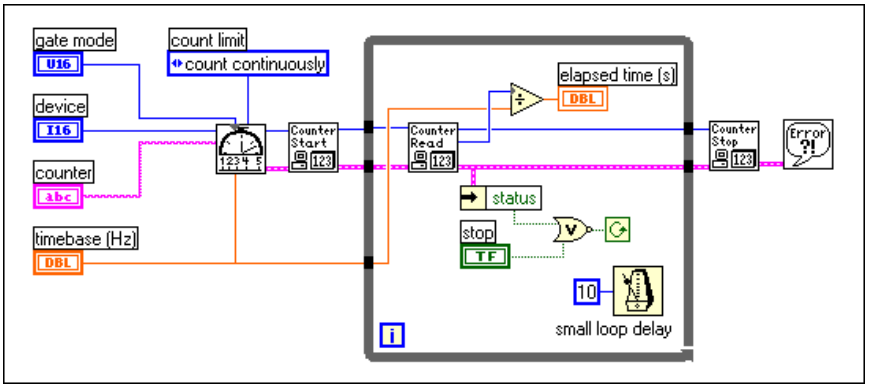



**Figure 27-11.** Diagram of Count Time-Int (DAQ-STC) VI

This example uses the Intermediate VIs Event or Time Counter Config, Counter Start, Counter Read, and Counter Stop. The Event or Time Counter Config VI configures **counter** to count the number of rising edges of a known internal timebase. The Counter Start VI begins the counting operation for **counter**. The Counter Read VI returns the count until you click the **STOP** button or an error occurs. The count value is divided by the **timebase** to determine the **elapsed time**. Finally, the Counter Stop VI stops the counter operation. You do not need to make any external connections, but you can optionally gate **counter** with a pulse to control when it starts and stops timing. To do this, wire your pulse to the GATE of **counter**, and choose the appropriate **gate mode** from the front panel menu. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

# Am9513

Figure 27-12 shows the Count Time-Easy (9513) VI located in `labview\examples\daq\counter\Am9513.llb`. This example uses the Easy VI, Count Events or Time, which can be found in **Functions»Data Acquisition»Counter**.

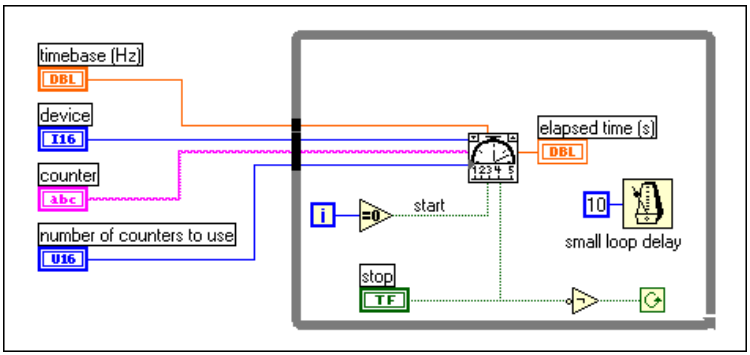



**Figure 27-12.** Diagram of Count Time-Easy (9315) VI

This VI initiates the counter to count the number of rising edges of a known internal timebase at the SOURCE of **counter**. The Count Events or Time VI takes care of dividing the count by the timebase frequency to determine the **elapsed time**. The counter continues timing until you click the **STOP** button. You do not need to make any external connections if the **number of counters to use** menu is set to **one counter (16-bits)**. If you **set the number of counters to use** menu to **two counters (32-bits)**, you must externally wire the OUT of **counter** to the SOURCE of **counter+1**. The length of time that can be counted depends on the maximum count of the counter(s) and the chosen **timebase**. For example, the 65,535 (16-bit) count of the Am9513 and a timebase of 1 MHz can count time for 65 ms. Using the 100 Hz timebase and two counters (32-bits), you can count time for over a year. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

If you need more control over when your elapsed timing begins and ends, use the Intermediate VIs instead of the Easy VIs. Figure 27-13 shows the Count Time-Int (9513) VI located in `labview\examples\daq\counter\Am9513.llb`.

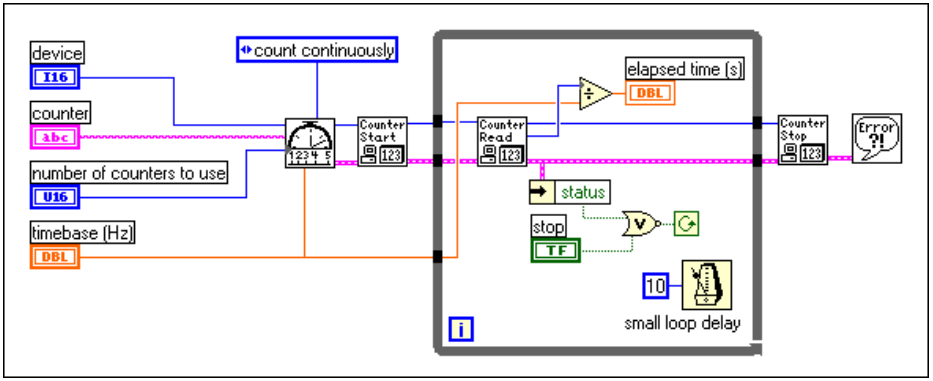



**Figure 27-13.** Diagram of Count Time-Int (9513) VI

This example uses the Intermediate VIs Event or Time Counter Config, Counter Start, Counter Read, and Counter Stop. The Event or Time Counter Config VI configures **counter** to count the number of rising edges of a known internal timebase. The Counter Start VI begins the counting operation for **counter**. The Counter Read VI returns the count until you click the **STOP** button or an error occurs. The count value is divided by the **timebase** to determine the **elapsed time**. Finally, the Counter Stop VI stops the counter operation. You can optionally gate **counter** with a pulse to control when it starts and stops timing. To do this, wire your pulse to the GATE of **counter**, and choose the appropriate **gate mode** from the front panel menu. Additionally, you can cascade two counters by choosing **two counters (32-bits)** in the **number of counters to use** menu. This extends your elapsed time range. You must also wire the OUT of **counter** to the SOURCE of **counter+1** for this increased range. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

## 8253/54

Figure 27-14 shows the Count Time (8253) VI located in `labview\examples\daq\counter\8253.llb`. This example uses the ICTR Control-Int VI, which can be found in **Functions» Data Acquisition»Counter»Intermediate Counter**.

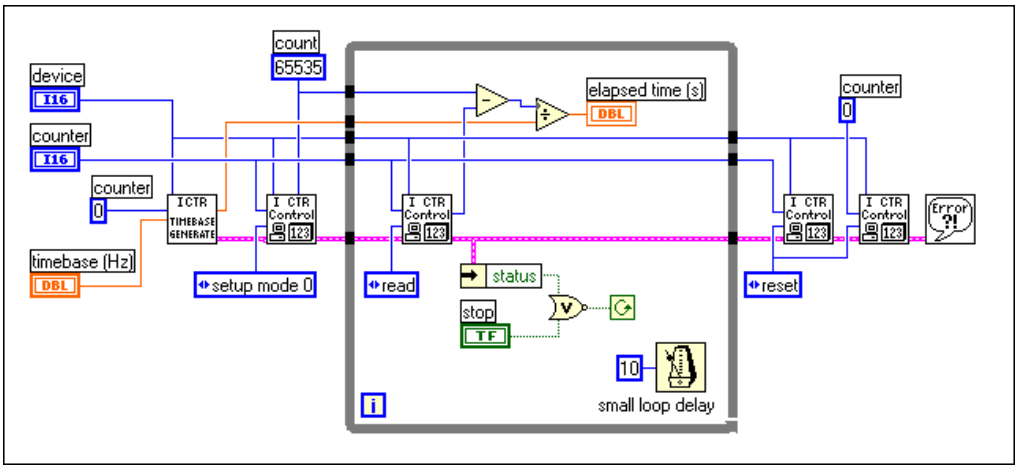



**Figure 27-14.** Diagram of Count Time (8253) VI

This VI initiates the counter to count the number of rising edges of a TTL timebase at the CLK of **counter**. **Counter 0** creates the timebase. Looking at the diagram, the Timebase Generator (8253) VI sets up **Counter 0** to generate a timebase by dividing down its internal timebase. The first call to ICTR Control loads the count register and sets up **counter** to count down. Inside the While Loop, ICTR Control reads the count, which is divided by the actual timebase frequency to determine the **elapsed time**. The elapsed time increments until you click the **STOP** button or an error occurs. The last two calls to ICTR Control reset **Counter 0** and **counter**. Remember that you must externally wire the OUT of **Counter 0** to the CLK of **counter**. You can optionally gate **counter** with a pulse to control when it starts and stops timing. To do this, wire your pulse to the GATE of **counter**. For a complete description of this example, refer to the information found in **Windows»Show VI Info**.

# 28

# Dividing Frequencies

Dividing TTL frequencies is useful if you want to use an internal timebase and the frequency you need does not exist. You can divide an existing internal frequency to get what you need. You can also divide the frequency of an external TTL signal. Frequency division results in a pulse or pulse train from a counter for every *N* cycles of an internal or external source. Counters can only decrease (divide down) the frequency of the source signal. The resulting frequency is equal to the input frequency divided by *N* (timebase divisor). *N* must be an integer number greater than 1. Performing frequency division on an internal signal is called a *down counter*. Frequency division on an external signal is called a *signal divider*. Figure 28-1 shows typical wiring for frequency division.

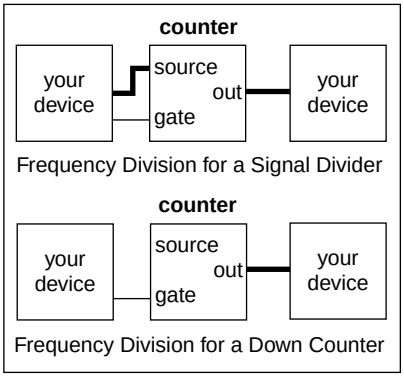



**Figure 28-1.** Wiring Your Counters for Frequency Division

## TIO-ASIC, DAQ-STC, Am9513

Figure 28-2 shows an example of a signal divider. It uses the Intermediate counter VIs Down Counter or Divide Config, Counter Start, and Counter Stop.

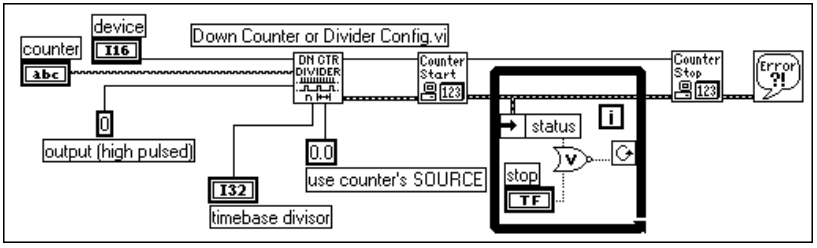



**Figure 28-2.** Programming a Single Divider for Frequency Division

The Down Counter or Divide Config VI configures the specified counter to divide the SOURCE signal by the **timebase divisor** value and output a signal when the counter reaches its terminal count (TC). Using Down Counter or Divide Config VI, you can configure the type of output to be pulse or toggled. The diagram above outputs a high pulse lasting one cycle of the source signal once the counter reaches its TC. For more information on the different types of signal outputs, refer to the Down Counter or Divide Config VI description in Chapter 27, *Intermediate Counter VIs*, of the *LabVIEW Function and VI Reference Manual*, or the LabVIEW *Online Reference*, available by selecting **Help»Online Reference**. The previous block diagram counts the rising edges of the SOURCE signal, the default value of the **source edge** input. To locate the inputs and outputs on this VI, use the **Help** window. Open this window by choosing **Help»Show Help**.

The Counter Start VI tells the counter to start counting the SOURCE signal edges. The counter stops the frequency division only when you click the **STOP** button. The Counter Stop VI stops the counter immediately and clears the count register. It is a good idea to always check your errors at the end of an operation to see if the operation was successful.

You can alter the Down Counter or Divide Config VI to create a down counter. To do this, change the timebase value from `0.0` (external SOURCE) to a frequency available on your counter. With the Am9513 chip, you can choose timebases of 1 MHz, 100 kHz, 10 kHz, 1 kHz, and 100 Hz. With the DAQ-STC chip, you can choose timebases of 20 MHz and 100 kHz.

Instead of triggering frequency division for signal dividers and down counters by software, as previously described, you can trigger using the GATE signal. You can trigger while the GATE signal is high, low, or on the rising or falling edge. For more information, refer to the Down Counter or Divide Config VI description in Chapter 27, *Intermediate Counter VIs*, of the *LabVIEW Function and VI Reference Manual*, or the LabVIEW *Online Reference*, available by selecting **Help»Online Reference**.

## 8253/54

To divide a frequency with the 8253/54 counter chip, use the example Cont Pulse Train (8253) VI located in `labview\examples\daq\8253.llb`. This example is explained in Chapter 24, *Generating a Square Pulse or Pulse Trains*, in the *Generating a Pulse Train* section. For a complete description of this example, refer to the information found in **Windows»VI Info**.

# Part VII

# Debugging Your Data Acquisition Application

This section contains an explanation of ways you can debug your data acquisition application to make sure your application is accurate and runs smoothly.

Part VII, *Debugging Your Data Acquisition Application*, contains Chapter 29, *Debugging Techniques*, which describes some tips to help determine why your VI is not working.

# 29

# Debugging Techniques

Is your VI not working as you expected? This chapter shows you some tips to help learn why your VI is not working. With LabVIEW DAQ applications, you might find errors in hardware connections, software configuration, or VI construction. The goal of this chapter is to help you narrow down where the problem is in your program flow.

## Hardware Connection Errors

When no error occurs but the data is not what you expected, you might want to check your hardware connections and jumper settings. For example, if you have an analog input application, make sure your signals are properly grounded. For more information on analog input configuration issues, refer to Chapter 5, *Things You Should Know about Analog Input*.

For SCXI modules, verify that gain jumpers are set up properly. To verify how a DAQ device gets set to a certain gain (or limit setting as noted in the software), refer to Chapter 3, *Basic LabVIEW Data Acquisition Concepts*. Another common SCXI hardware error is using digital lines on your DAQ device that are reserved for communication with the SCXI modules.

To test that your hardware has not been damaged, connect a known voltage to the channels you are using. To check the location of any hardware connections, refer to your hardware user manual.

## Software Configuration Errors

As you check hardware connections, verify that the NI-DAQ software configuration reflects your hardware setup. For possible difficulties with software configuration, read Chapter 2, *Installing and Configuring Your Data Acquisition Hardware*, the chapter of this manual that describes your specific application, or the *NI-DAQ User Manual*.

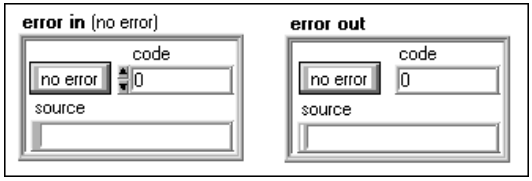
**(Windows)** In the NI-DAQ Configuration Utility, you can use the NI-DAQ Test Panels to verify that your device is operating properly. Refer to the NI-DAQ Configuration Utility online help or Measurement & Automation Explorer online help for more details.

# VI Construction Errors

The various sections below describe methods to find problems with VI construction. All the techniques described can be used by themselves or in conjunction with one another.

## Error Handling

The best way to determine if your application executed without an error is to use one of the error handler VIs in your application. The Error Handler VIs are located in **Functions»Time & Dialog**. You can only use these VIs with Intermediate and Advanced VIs. Easy I/O VIs already include error handling capabilities within each VI. Each Intermediate and Advanced VI has an error input and output clusters (named **error in** and **error out**, respectively). The error clusters contain a Boolean that indicates whether an error occurred, the error code for the error, and the name of the VI that returned the error. If **error in** indicates an error, the VI returns the same error information in **error out**, and does not perform any DAQ operations.




When you use any of the Intermediate or Advanced VIs in a While Loop, you should stop the loop if the status in the **error out** cluster reads TRUE. If you wire the error cluster to the General Error Handler VI or the Simple Error Handler VI, the VI deciphers the error information and describes the error to you. Figures 29-1 and 29-2 show how to wire a typical DAQ VI to an error handler.

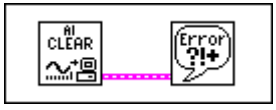



**Figure 29-1.** Error Checking Using the General Error Handler VI

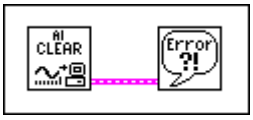
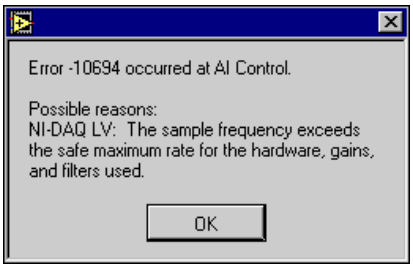



**Figure 29-2.** Error Checking Using the Simple Error Handler VI

The following figure shows an example of the dialog box the Error Handler VIs display if an error occurs.


Error -10694 occurred at AI Control.
Possible reasons:
NI-DAQ LV: The sample frequency exceeds the safe maximum rate for the hardware, gains, and filters used.
OK

Refer to the *LabVIEW Function and VI Reference Manual* or the LabVIEW *Online Reference*, available by selecting **Help»Online Reference**, for more information on the error handler VIs.

## Single-Stepping through a VI

Single-stepping through a VI allows you to execute one node at a time in the block diagram. A node can be subVIs, functions, structures, formula nodes, and attribute nodes. Refer to Chapter 2, *Creating VIs*, in the *LabVIEW User Manual*, and Chapter 4, *Executing and Debugging VIs and SubVIs*, in the *G Programming Reference Manual* for more information about single-stepping.

## Execution Highlighting

Execution highlighting (the light bulb button on the diagram) shows you how data passes from one node to another in your program. When you turn execution highlighting on, data movement is marked by bubbles moving along the wires. Refer to Chapter 2, *Creating VIs*, in the *LabVIEW User Manual*, and Chapter 4, *Executing and Debugging VIs and SubVIs*, in the *G Programming Reference Manual* for more information about execution highlighting.

## Using the Probe Tool

If your VI is producing questionable results, you may want to use the Probe tool to check intermediate values in a VI. The Probe tool will help you narrow down where the incorrect results are occurring. Refer to Chapter 2, *Creating VIs*, in the *LabVIEW User Manual* and Chapter 4, *Executing and Debugging VIs and SubVIs*, in the *G Programming Reference Manual* for more information on using the probe.

## Setting Breakpoints and Showing Advanced DAQ VIs

Once you have narrowed down the location of an error to a subVI, you can set a breakpoint on that subVI to cause VI execution to pause before executing the subVI. You can now see what values get passed in or are generated by the Advanced VIs, single-step through the subVI's execution, probe wires to see data, or change values of front panel controls. Refer to Chapter 2, *Creating VIs*, in the *LabVIEW User Manual* and Chapter 4, *Executing and Debugging VIs and SubVIs*, in the *G Programming Reference Manual* for more information on how to set a breakpoint.

# A

# LabVIEW Data Acquisition Common Questions

This appendix lists answers to questions frequently asked by LabVIEW users.

**Where is the best place to get up to speed quickly with data acquisition and LabVIEW?**

Read this manual and look at the `run_me.llb` examples, in `labview\examples\daq\run_me.llb`, included with LabVIEW. In Windows, run Measurement & Automation Explorer or the DAQ Channel Wizard and the DAQ Solution Wizard.

**What is the easiest way to address my AMUX-64T device with my MIO device?**

Set the number of AMUX devices used in Measurement & Automation Explorer (**Windows**) or `NI-DAQ` control panel (**Macintosh**). Then in the channel string inputs, specify the onboard channel. For example, with one AMUX-64T device, the channel string `0:1` will acquire data from AMUX channels 0 through 7, and so on.

**What are the advantages/disadvantages of reading AI Read's backlog rather than a fixed amount of data?**

Reading the backlog is guaranteed not to cause a synchronous wait for the data to arrive. However, it adds more delay until the data is processed (because the data was available on the last call) and it can require constant reallocation or size adjustments of the data acquisition read buffer in LabVIEW.

**What is the easiest way to verify that my device works and is acquiring data from my signals?**

Run one of the examples in the `labview\examples\daq` folder or run the test panel for your device in Measurement & Automation Explorer.

**How can I tell when a continuous data acquisition operation does not have enough buffer capacity?**

The scan backlog rises with time, either steadily or in jumps, or takes a long time to drop to normal after an interrupting activity like mouse movement. If you can open another VI during the operation without receiving an overrun error you should have adequate buffer capacity.

**I want to group two or more ports using my DIO32, DIO24, or DIO-96 device, but I do not want to use handshaking. I just want to read one group of ports just once. How can I set it up in software?**

Use Easy I/O VIs (Write to Digital Port or Read from Digital Port) or Advanced Digital VIs (DIO Port Config, DIO Port Write or DIO Port Read), and set multiple ports in the port list. For Easy I/O VIs, you can specify up to four ports in the port list. Whatever data you try to output to each port of your "group" will correspond to each element of the data array. This also applies for input.

**I want to use the OUT1, OUT2, OUT3 and IN1, IN2, IN3 pins on my DIO-32F device. How do I address those pins using the Easy I/O Digital VIs in LabVIEW?**

These output and inputs pins are addressed together as port 4. OUT1 and IN1 are referred to as bit 0, OUT2 and IN2 are referred to as bit 1, and OUT3 and IN3 are referred to as bit 2. Only the NB-DIO-32F has three pins for each direction. If you use the Write To Digital Port VI, you will output on the OUT pins. If you use the Read From Digital Port VI, you will input from the IN pins.

**I want to be able to write up to four lines on the digital port on my jumpered MIO (non E Series) device while also reading in four lines of digital data on the remaining free digital lines. How do I do this?**

Use the DIO Port Config VI twice—once to configure four lines for output and once more to configure four lines for input. Now call the DIO Port Write VI or the DIO Port Read VI for the appropriate lines. Avoid calling the Easy I/O VIs for digital I/O, as they reconfigure the port direction each time the VI is called.

**I want to use a TTL digital trigger pulse to start data acquisition on my DAQ device. I noticed there are two types of triggers: Digital Trigger A and Digital Trigger A&B. Which digital trigger setting should I use and where should I connect the signal?**

You should use Digital Trigger A, which stands for "first trigger," to start a data acquisition. Digital Trigger B, which stands for "second trigger," should only be used if you are doing both a start *and* a stop trigger for your data acquisition. Connect your trigger signal to either STARTTRIG* (pin 38) if you are using an AT-MIO-16, AT-MIO-16D, NB-MIO-16X, or EXTTRIG* or DTRIG for any other device that has that pin. If you are using an E-series device, you can select which PFI pin to connect to. If you do not specify the PFI pin, it uses the defaults as the PFI pin names suggest, for example, PFI0/TRIG1. The only analog input devices on which you cannot do a digital trigger are the LPM devices, DAQCard-700, DAQCard-500, and the 516 devices. Refer to the AI Trigger Config description in Chapter 18, *Advanced Analog Input VIs*, in the *LabVIEW Function and VI Reference Manual*, or the LabVIEW *Online Reference*, available by selecting **Help»Online Reference**, for more information on the use of digital triggers on your DAQ device.

**Note** The NB-MIO-16 has an EXTTRIG* pin but cannot support start and stop triggering.

**When are the data acquisition devices initialized?**

All data acquisition devices are initialized automatically when the first DAQ VI is loaded in on a diagram when you start LabVIEW. You can also initialize a particular device by calling the Device Reset VI.

# B

# Technical Support Resources

This appendix describes the comprehensive resources available to you in the Technical Support section of the National Instruments Web site and provides technical support telephone numbers for you to use if you have trouble connecting to our Web site or if you do not have internet access.

## NI Web Support

To provide you with immediate answers and solutions 24 hours a day, 365 days a year, National Instruments maintains extensive online technical support resources. They are available to you at no cost, are updated daily, and can be found in the Technical Support section of our Web site at `www.natinst.com/support`.

### Online Problem-Solving and Diagnostic Resources

- **KnowledgeBase**—A searchable database containing thousands of frequently asked questions (FAQs) and their corresponding answers or solutions, including special sections devoted to our newest products. The database is updated daily in response to new customer experiences and feedback.
- **Troubleshooting Wizards**—Step-by-step guides lead you through common problems and answer questions about our entire product line. Wizards include screen shots that illustrate the steps being described and provide detailed information ranging from simple getting started instructions to advanced topics.
- **Product Manuals**—A comprehensive, searchable library of the latest editions of National Instruments hardware and software product manuals.
- **Hardware Reference Database**—A searchable database containing brief hardware descriptions, mechanical drawings, and helpful images of jumper settings and connector pinouts.
- **Application Notes**—A library with more than 100 short papers addressing specific topics such as creating and calling DLLs, developing your own instrument driver software, and porting applications between platforms and operating systems.

## Software-Related Resources

- **Instrument Driver Network**—A library with hundreds of instrument drivers for control of standalone instruments via GPIB, VXI, or serial interfaces. You also can submit a request for a particular instrument driver if it does not already appear in the library.
- **Example Programs Database**—A database with numerous, non-shipping example programs for National Instruments programming environments. You can use them to complement the example programs that are already included with National Instruments products.
- **Software Library**—A library with updates and patches to application software, links to the latest versions of driver software for National Instruments hardware products, and utility routines.

# Worldwide Support

National Instruments has offices located around the globe. Many branch offices maintain a Web site to provide information on local services. You can access these Web sites from `www.natinst.com/worldwide`.

If you have trouble connecting to our Web site, please contact your local National Instruments office or the source from which you purchased your National Instruments product(s) to obtain support.

For telephone support in the United States, dial 512 795 8248. For telephone support outside the United States, contact your local branch office:

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Brazil 011 284 5011, Canada (Calgary) 403 274 9391,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Greece 30 1 42 96 427
Hong Kong 2645 3186, India 91805275406, Israel 03 6120092,
Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico (D.F.) 5 280 7625, Mexico (Monterrey) 8 357 7695,
Netherlands 0348 433466, Norway 32 27 73 00, Singapore 2265886,
Spain (Barcelona) 93 582 0251, Spain (Madrid) 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51,
Taiwan 02 2377 1200, United Kingdom 01635 523545

# Glossary

| Prefix | Meaning | Value |
|---|---|---|
| k- | kilo- | $10^3$ |
| M- | mega- | $10^6$ |
| m- | milli- | $10^{-3}$ |
| µ- | micro- | $10^{-6}$ |
| n- | nano- | $10^{-9}$ |

## Numbers/Symbols

1D — One-dimensional.

2D — Two-dimensional.

## A

A — Amperes.

AC — Alternating current.

A/D — Analog-to-digital.

ADC — Analog-to-digital converter. An electronic device, often an integrated circuit, that converts an analog voltage to a digital number.

ADC resolution — The resolution of the ADC, which is measured in bits. An ADC with 16 bits has a higher resolution, and thus a higher degree of accuracy than a 12-bit ADC.

AI — Analog input.

AI device — An analog input device that has AI in its name, such as the NEC-AI-16E-4.

AIGND — The analog input ground pin on a DAQ device.

| | |
|---|---|
| amplification | A type of signal conditioning that improves accuracy in the resulting digitized signal and reduces noise. |
| Am9513-based devices | Devices with an Am9513 counter/timer chip. These devices include the NB-MIO-16, NB-MIO-16X, NB-A2000, NB-TIO-10, and NB-DMA2800 on the Macintosh; and the AT-MIO-16, AT-MIO-16F-5, AT-MIO-16X, AT-MIO-16D, and AT-MIO-64F-5, and PC-TIO-10 in Windows. |
| AMUX devices | *See* analog multiplexers. |
| analog input group | A collection of analog input channels. You can associate each group with its own clock rates, trigger and buffer configurations, and so on. A channel cannot belong to more than one group.<br><br>Because each device has one ADC, only one group can be active at any given time. That is, once a control VI starts a timed acquisition with group *n*, subsequent control and read calls must also refer to group *n*. You use the task ID to refer to the group. |
| analog multiplexer | Devices that increase the number of measurement channels while still using a single instrumentation amplifier. *Also called* AMUX devices. |
| analog output group | A collection of analog output channels. You can associate each group with its own clock rates, buffer configurations, and so on. A channel cannot belong to more than one group. |
| analog trigger | A trigger that occurs at a user-selected level and slope on an incoming analog signal. Triggering can be set to occur at a specified voltage on either an increasing or a decreasing signal (positive or negative slope). |
| AO | Analog output. |
| array | Ordered, indexed set of data elements of the same type. |

## B

| | |
|---|---|
| BCD | Binary-coded decimal. |
| bipolar | A signal range that includes both positive and negative values (for example, –5 to 5 V). |
| buffer | Temporary storage for acquired or generated data. |

# C

| | |
|---|---|
| cascading | Process of extending the counting range of a counter chip by connecting to the next higher counter. |
| channel | Pin or wire lead to which you apply or from which you read the analog or digital signal. Analog signals can be single-ended or differential. For digital signals, you group channels to form ports. Ports usually consist of either four or eight digital channels. |
| channel clock | The clock controlling the time interval between individual channel sampling within a scan. Boards with simultaneous sampling do not have this clock. |
| channel name | A unique name given to a channel configuration in the Data Neighborhood in the Measurement and Automation Explorer |
| circular-buffered I/O | Input/output operation that reads or writes more data points than can fit in the buffer. When LabVIEW reaches the end of the buffer, LabVIEW returns to the beginning of the buffer and continues to transfer data. |
| clock | Hardware component that controls timing for reading from or writing to groups. |
| cluster | A set of ordered, unindexed data elements of any data type including numeric, Boolean, string, array, or cluster. The elements must be all controls or all indicators. |
| code width | The smallest detectable change in an input voltage of a DAQ device. |
| column-major order | A way to organize the data in a 2D array by columns. |
| common-mode voltage | Any voltage present at the instrumentation amplifier inputs with respect to amplifier ground. |
| conditional retrieval | A method of triggering in which you to simulate an analog trigger using software. *Also called* software triggering. |
| configuration utility | Refers to a utility that allows you to configure your hardware. |
| conversion device | Device that transforms a signal from one form to another. For example, analog-to-digital converters (ADCs) for analog input, digital-to-analog converters (DACs) for analog output, digital input or output ports, and counter/timers are conversion devices. |

| | |
|---|---|
| counter/timer group | A collection of counter/timer channels. You can use this type of group for simultaneous operation of multiple counter/timers. |
| coupling | The manner in which a signal is connected from one location to another. |

## D

| | |
|---|---|
| D/A | Digital-to-analog. |
| DAC | Digital-to-analog converter. An electronic device, often an integrated circuit, that converts a digital number into a corresponding analog voltage or current. |
| DAQ Channel Wizard | Utility that guides you through naming and configuring your DAQ analog and digital channels. |
| DAQ Solution Wizard | Utility that guides you through specifying your DAQ application, from which it provides a custom DAQ solution. |
| DAQ-STC | Data Acquisition System Timing Controller. |
| data acquisition | Process of acquiring data, typically from A/D or digital input plug-in boards. |
| data flow | Programming system consisting of executable nodes in which nodes execute only when they have received all required input data and produce output automatically when they have executed. LabVIEW is a dataflow system. |
| default input | The default value of a front panel control. |
| default load area | One of three parts of the SCXI EEPROM. The default load area is where LabVIEW automatically looks to load calibration constants the first time you access an SCXI module. When the module is shipped, this area contains a copy of the factory calibration constants. The other EEPROM areas are the factory area and the user area. |
| default setting | A default parameter value recorded in the driver. In many cases, the default input of a control is a certain value (often 0) that means *use the current default setting*. For example, the default input for a parameter may be *do not change current setting*, and the default setting may be *no AMUX-64T boards*. If you do change the value of such a parameter, the new value becomes the new setting. You can set default settings for some parameters in the configuration utility. |

| | |
|---|---|
| device | A DAQ device inside your computer or attached directly to your computer through a parallel port. Plug-in boards, PC cards, and devices such as the DAQPad-1200, which connects to your computer's parallel port, are all examples of DAQ devices. SCXI modules are distinct from devices, with the exception of the SCXI-1200, which is a hybrid. |
| device number | The slot number or device ID number assigned to the device when you configured it. |
| DIFF | Differential. A differential input is an analog input consisting of two terminals, both of which are isolated from computer ground and whose difference you measure. |
| differential measurement system | A way you can configure your device to read signals, in which you do not need to connect either input to a fixed reference, such as earth or a building ground. |
| digital input group | A collection of digital input ports. You can associate each group with its own clock rates, handshaking modes, buffer configurations, and so on. A port cannot belong to more than one group. |
| digital output group | A collection of digital output ports. You can associate each group with its own clock rates, handshaking modes, buffer configurations, and so forth. A port cannot belong to more than one group. |
| digital trigger | A TTL signal that you can use to start or stop a buffered data acquisition operation, such as buffered analog input or buffered analog output. |
| DIO devices | Refers to all devices with the letters DIO in their name, unless otherwise noted. |
| DIP | Dual Inline Package. |
| dithering | The addition of Gaussian noise to an analog input signal. By applying dithering and then averaging the input data, you can effectively increase the resolution by another one-half bit. |
| DLL | Dynamic Link Library. |
| DMA | Direct Memory Access. A method by which you can transfer data to computer memory from a device or memory on the bus (or from computer memory to a device) while the processor does something else. DMA is the fastest method of transferring data to or from computer memory. |
| down counter | Performing frequency division on an internal signal. |

| | |
|---|---|
| driver | Software that controls a specific hardware device, such as a DAQ device. |

## E

| | |
|---|---|
| E Series MIO device | Boards, such as the PCI-MIO-16E-1 and the AT-MIO-16E-2 that use the MITE chip (on PCI boards for bus mastering), the DAQ-PnP chip for Plug and Play configuration, the DAQ-STC chip for instrumentation class counting and timing, and the NI-PGIA for high accuracy analog input measurements. |
| EEPROM | Electrically erasable programmable read-only memory. Read-only memory that you can erase with an electrical signal and reprogram. |
| EISA | Extended Industry Standard Architecture. |
| event | The condition or state of an analog or digital signal. |
| external trigger | A voltage pulse from an external source that triggers an event such as A/D conversion. |

## F

| | |
|---|---|
| factory area | One of three parts of the SCXI EEPROM. The factory area contains factory-set calibration constants. This area is read-only. The other EEPROM areas are the default load area and the user area. |
| FIFO | A first-in-first-out memory buffer. In a FIFO, the first data stored is the first data sent to the acceptor. |
| filtering | A type of signal conditioning that allows you to filter unwanted signals from the signal you are trying to measure. |
| floating signal sources | Signal sources with voltage signals that are not connected to an absolute reference or system ground. Some common example of floating signal sources are batteries, transformers, or thermocouples. *Also called* nonreferenced signal sources. |

## G

| | |
|---|---|
| gain | The amplification or attenuation of a signal. |
| GATE input pin | A counter input pin that controls when counting in your application occurs. |

| | |
|---|---|
| grounded measurement system | *See* referenced single-ended measurement system. |
| grounded signal sources | Signal sources with voltage signals that are referenced to a system ground, such as earth or a building ground. *Also called* referenced signal sources. |
| group | A collection of input or output channels or ports that you define. Groups can contain analog input, analog output, digital input, digital output, or counter/timer channels. A group can contain only one type of channel, however. You use a task ID number to refer to a group after you create it. You can define up to 16 groups at one time. |

## H

| | |
|---|---|
| handshaked digital I/O | A type of digital acquisition/generation where a device or module accepts or transfers data after a digital pulse has been received. *Also called* latched digital I/O. |
| hardware triggering | A form of triggering where you set the start time of an acquisition and gather data at a known position in time relative to a trigger signal. |
| hex | Hexadecimal. |
| Hz | Hertz. The number of scans read or updates written per second. |

## I

| | |
|---|---|
| IEEE | Institute of Electrical and Electronic Engineers. |
| immediate digital I/O | A type of digital acquisition/generation where LabVIEW updates the digital lines or port states immediately or returns the digital value of an input line. *Also called* nonlatched digital I/O. |
| input limits | The upper and lower voltage inputs for a channel. You must use a pair of numbers to express the input limits. The VIs can infer the input limits from the input range, input polarity, and input gain(s). Similarly, if you wire the input limits, range, and polarity, the VIs can infer the onboard gains when you do not use SCXI. |

| | |
|---|---|
| input range | The difference between the maximum and minimum voltages an analog input channel can measure at a gain of 1. The input range is a scalar value, not a pair of numbers. By itself the input range does not uniquely determine the upper and lower voltage limits. An input range of 10 V could mean an upper limit of +10 V and a lower of 0 V or an upper limit of +5 V and a lower limit of –5 V.<br><br>The combination of input range, polarity, and gain determines the input limits of an analog input channel. For some boards, jumpers set the input range and polarity, while you can program them for other boards. Most boards have programmable gains. When you use SCXI modules, you also need their gains to determine the input limits. |
| interrupt | A signal indicating that the central processing unit should suspend its current task to service a designated activity. |
| interval scanning | Scanning method where there is a longer interval between scans than there is between individual channels comprising a scan. |
| I/O | Input/output. The transfer of data to or from a computer system involving communications channels, operator interface devices, and/or data acquisition and control interfaces. |
| ISA | Industry Standard Architecture. |
| isolation | A type of signal conditioning in which you isolate the transducer signals from the computer for safety purposes. This protects you and your computer from large voltage spikes and makes sure the measurements from the DAQ device are not affected by differences in ground potentials. |

## K

| | |
|---|---|
| Kwords | 1,024 words of memory. |

## L

| | |
|---|---|
| Lab/1200 devices | Boards, such as the Lab-PC-1200 and the DAQCard-1200, which use the 8253 type counter/timer chip. |
| LabVIEW | Laboratory Virtual Instrument Engineering Workbench. |

| | |
|---|---|
| latched digital I/O | A type of digital acquisition/generation where a device or module accepts or transfers data after a digital pulse has been received. *Also called* handshaked digital I/O. |
| Legacy MIO device | Boards, such as the AT-MIO-16, which typically are configured with jumpers and switches and are not Plug and Play compatible. They also use the 9513 type counter/timer chip. |
| limit settings | The maximum and minimum voltages of the analog signals you are measuring or generating. |
| linearization | A type of signal conditioning in which LabVIEW linearizes the voltage levels from transducers, so the voltages can be scaled to measure physical phenomena. |
| LSB | Least Significant Bit. |

## M

| | |
|---|---|
| MB | Megabytes of memory. 1 MB is equal to 1,024 KB. |
| memory buffer | *See* buffer. |
| Measurement & Automation Explorer | The standard National Instruments hardware configuration and diagnostic environment for Windows. |
| multibuffered I/O | Input operation for which you allocate more than one memory buffer so you can read and process data from one buffer while the acquisition fills another. |
| multiplexed mode | An SCXI operating mode in which analog input channels are multiplexed into one module output so that your cabled DAQ device has access to the module's multiplexed output as well as the outputs on all other multiplexed modules in the chassis through the SCXIbus. *Also called* serial mode. |
| multiplexer | A set of semiconductor or electromechanical switches with a common output that can select one of a number of input signals and that you commonly use to increase the number of signals measured by one ADC. |

## N

| | |
|---|---|
| NB | NuBus. |
| NI-DAQ | The NI-DAQ software system, which contains a driver, a configuration utility, a suite of electronic documentation, and a rich set of examples. |
| nodes | Execution elements of a block diagram consisting of functions, structures, and subVIs. |
| nonlatched digital I/O | A type of digital acquisition/generation where LabVIEW updates the digital lines or port states immediately or returns the digital value of an input line. *Also called* immediate digital I/O. |
| nonreferenced signal sources | Signal sources with voltage signals that are not connected to an absolute reference or system ground. *Also called* floating signal sources. Some common examples of nonreferenced signal sources are batteries, transformers, or thermocouples. |
| Nonreferenced single-ended (NRSE) measurement system | All measurements are made with respect to a common reference, but the voltage at this reference can vary with respect to the measurement system ground. |
| NRSE | Nonreferenced single-ended. |

## O

| | |
|---|---|
| onboard channels | Channels provided by the plug-in data acquisition device. |
| OUT pin | A counter output pin where the counter can generate various TTL pulse waveforms. |
| output limits | The upper and lower voltage or current outputs for an analog output channel. The output limits determine the polarity and voltage reference settings for a device. |

# P

| | |
|---|---|
| parallel mode | A type of SCXI operating mode in which the module sends each of its input channels directly to a separate analog input channel of the device to the module. |
| pattern generation | A type of handshaked (latched) digital I/O in which internal counters generate the handshaked signal, which in turn initiates a digital transfer. Because counters output digital pulses at a constant rate, this means you can generate and retrieve patterns at a constant rate because the handshaked signal is produced at a constant rate. |
| PCI | Peripheral Component Interconnect. An industry-standard, high-speed databus. |
| PGIA | Programmable Gain Instrumentation Amplifier. |
| Plug and Play devices | Devices that do not require DIP switches or jumpers to configure resources on the devices. *Also called* switchless devices. |
| postriggering | The technique you use on a DAQ device to acquire a programmed number of samples after trigger conditions are met. |
| pretriggering | The technique you use on a DAQ device to keep a continuous buffer filled with data, so that when the trigger conditions are met, the sample includes the data leading up to the trigger condition. |
| pulse trains | Multiple pulses. |
| pulsed output | A form of counter signal generation by which a pulse is outputted when a counter reaches a certain value. |
| PXI | PCI eXtensions for Instrumentation. A modular, computer-based instrumentation platform. |

# R

| | |
|---|---|
| read mark | Points to the scan at which a read operation begins. Analogous to a file I/O pointer, the read mark moves every time you read data from an input buffer. After the read is finished, the read mark points to the next unread scan. Because multiple buffers are possible, you need both the buffer number and the scan number to express the position of the read mark. |
| read mode | Indicates one of the four reference marks within an input buffer that provides the reference point for the read. This reference can be the read mark, the beginning of the buffer, the most recently acquired data, or the trigger position. |
| referenced signal sources | Signal sources with voltage signals that are referenced to a system ground, such as earth or a building ground. *Also called* grounded signal sources. |
| referenced single-ended (RSE) measurement system | All measurements are made with respect to a common reference or a ground. *Also called* a grounded measurement system. |
| RMS | Root Mean Square. |
| row-major order | A way to organize the data in a 2D array by rows. |
| RSE | Referenced Single-Ended. |
| RTD | Resistance Temperature Detector. A temperature-sensing device whose resistance increases with increases in temperature. |
| RTSI | Real-Time System Integration bus. The National Instruments timing bus that interconnects data acquisition boards directly, by means of connectors on top of the boards, for precise synchronization of functions. |

# S

| | |
|---|---|
| s | Seconds. |
| sample | A single analog or digital input or output data point. |
| sample counter | The clock that counts the output of the channel clock, in other words, the number of samples taken. On boards with simultaneous sampling, this counter counts the output of the scan clock and hence the number of scans. |

| | |
|---|---|
| scan | One or more analog or digital input samples. Typically, the number of input samples in a scan is equal to the number of channels in the input group. For example, one pulse from the scan clock produces one scan that acquires one new sample from every analog input channel in the group. |
| scan clock | The clock controlling the time interval between scans. On boards with interval scanning support (for example, the AT-MIO-16F-5), this clock gates the channel clock on and off. On boards with simultaneous sampling (for example, the EISA-A2000), this clock clocks the track-and-hold circuitry. |
| scan rate | The number of times (or scans) per second that LabVIEW acquires data from channels. For example, at a scan rate of 10Hz, LabVIEW samples each channel in a group 10 times per second. |
| scan width | The number of channels in the channel list or number of ports in the port list you use to configure an analog or digital input group. |
| SCXI | Signal Conditioning eXtensions for Instrumentation. The National Instruments product line for conditioning low-level signals within an external chassis near sensors, so only high-level signals in a noisy environment are sent to data acquisition boards. |
| settling time | The amount of time required for a voltage to reach its final value within specified limits. |
| signal conditioning | The manipulation of signals to prepare them for digitizing. |
| signal divider | Performing frequency division on an external signal. |
| simple-buffered I/O | Input/output operation that uses a single memory buffer big enough for all of your data. LabVIEW transfers data into or out of this buffer at the specified rate, beginning at the start of the buffer and stopping at the end of the buffer. You use simple buffered I/O when you acquire small amounts of data relative to memory constraints. |
| single-ended inputs | Analog inputs that you measure with respect to a common ground. |
| software trigger | A programmed event that triggers an event such as data acquisition. |
| software triggering | A method of triggering in which you to simulate an analog trigger using software. *Also called* conditional retrieval. |
| SOURCE input pin | An counter input pin where the counter counts the signal transitions. |

STC — System Timing Controller.

strain gauge — A thin conductor, which is attached to a material, that detects stress or vibrations in that material.

subVI — VI used in the block diagram of another VI; comparable to a subroutine.

switchless device — Devices that do not require DIP switches or jumpers to configure resources on the devices. *Also called* Plug and Play devices.

syntax — The set of rules to which statements must conform in a particular programming language.

## T

task — A timed I/O operation using a particular group. *See* task ID.

task ID — A number generated by LabVIEW, which identifies to the NI-DAQ drive the task at hand.

The following table gives the function code definitions.

.

| Function Code | I/O Operation |
|---|---|
| 1 | analog input |
| 2 | analog output |
| 3 | digital port I/O |
| 4 | digital group I/O |
| 5 | counter/timer I/O |

TC — Terminal count. The highest value of a counter.

TIO-ASIC — Timing I/O Application Specific Integrated Circuit. Found on 660*x* devices.

toggled output — A form of counter signal generation by which the output changes the state of the output signal from high to low, or low to high when the counter reaches a certain value.

top-level VI — VI at the top of the VI hierarchy. This term is used to distinguish the VI from its subVIs.

track-and-hold — A circuit that tracks an analog voltage and holds the value on command.

transducer excitation — A type of signal conditioning that uses external voltages and currents to excite the circuitry of a signal conditioning system into measuring physical phenomena.

trigger — Any event that causes or starts some form of data capture.

TTL — Transistor-Transistor Logic

## U

unipolar — A signal range that is either always positive or negative, but never both (for example 0 to 10 V, *not* –10 to 10 V).

update — One or more analog or digital output samples. Typically, the number of output samples in an update is equal to the number of channels in the output group. For example, one pulse from the update clock produces one update that sends one new sample to every analog output channel in the group.

update rate — The number of output updates per second.

update width — The number of channels in the channel list or number of ports in the port list you use to configure an analog or digital output group.

user area — One of three parts of the SCXI EEPROM. The user area is where you store calibration constants that you calculate using the SCXI Cal Constants VI. If you want LabVIEW to load your constants automatically, you can put a copy of your constants in the default load area. The other EEPROM areas are the factory area and the default load area.

## V

V — Volts.

VDC — Volts, Direct Current.

VI — Virtual Instrument. A LabVIEW program; so-called because it models the appearance and function of a physical instrument.

$V_{ref}$ — Voltage reference.

# W

| | |
|---|---|
| waveform | Multiple voltage readings taken at a specific sampling rate. |
| wire | Data path between nodes. |
| write mark | Points to the update at which a write operation begins. Analogous to a file I/O pointer, the write mark moves every time you write data into an output buffer. After the write is finished, the write mark points to the next update to be written. Because multiple buffers are possible, you need both the buffer number and the update number to express the position of the write mark. |

# Index

## Numbers


2D arrays, 3-14 to 3-16
8253/54 counter
  accuracy, 24-23
  continuous pulse train generation, 24-13 to 24-14
  description, 23-5 to 23-6
  determining pulse width, 25-5 to 25-6
  dividing frequencies, 28-3
  elapsed time counting, 27-12
  event counting, 27-7
  finite pulse train generation, 24-18 to 24-21
  frequency and period measurement
    high frequency signals, 26-7 to 26-8
    how and when to measure, 26-2
    low frequency signals, 26-10
  internal timebases with maximum pulse width measurements (table), 25-9
  single square pulse generation, 24-7 to 24-10
  square pulse generation, 24-4
  stopping counter generations, 24-24


## A


accessories, LabVIEW support for (table), 2-12
accessories folder, for DAQ examples, 3-2
ACK (Acknowledge Input) line, 17-2
Acquire & Proc N Scans-Trig example VI, 8-5, 8-9
Acquire & Process N Scans VI, 7-10
Acquire 1 Point from 1 Channel VI, 6-2
Acquire and Average VI, 21-7 to 21-8
Acquire N Scans Analog Hardware Trig example VI, 8-8 to 8-9
Acquire N Scans Analog Software Trig example VI, 8-12
Acquire N Scans Digital External Sample Trigger VI, 9-6
Acquire N Scans Digital Trig example VI, 8-4 to 8-5
Acquire N Scans example VI, 7-4, 7-6
Acquire N Scans-ExtChanClk example VI, 9-4, 9-7
Acquire N-Multi-Analog Hardware Trig example VI, 8-9
Acquire N-Multi-Digital Trig example VI, 8-5
Acquire N-Multi-Start example VI, 7-6 to 7-7
acquisition rate. *See* external control.
ADC
  limit settings effects (figure), 5-6
  measurement precision for various device ranges and limit settings (table), 5-8
  range effects (figure), 5-5
  resolution effects, 5-4
adjacent counters for counter chips (table), 27-2
Adjacent Counters VI, 26-6
Advanced VIs. *See also* VIs.
  analog output SCXI example, 21-17
  buffered pulse and period measurement, 25-8
  external control of channel clock, 9-4
  finite pulse train generation, 24-17
  non-buffered handshaking, 17-5
  overview, 3-5
  simple buffered handshaking, 17-8 to 17-9
AI Acquire Waveform VI, 7-2 to 7-3
AI Acquire Waveforms VI
  multiple-waveform acquisition, 7-3
  simple-buffered analog input with graphing, 7-5 to 7-6
AI Clear VI
  amplifier offset, 21-5
  hardware-timed analog I/O control loops, 6-9

multiple-waveform acquisition, 7-4
SCXI temperature measurement, 21-8
simultaneous buffered waveform acquisition and generation, 14-2
AI Clock Config VI
external control of channel clock, 9-4
external conversions, 9-4
retrieving channel clock setting, 20-5
scan clock control, 9-5 to 9-6, 9-8
AI Config VI
basic non-buffered application, 6-4
hardware-timed analog I/O control loops, 6-8
interchannel delay, 9-2
multiple-channel single-point analog input, 6-5
multiple-waveform acquisition, 7-4
one-point calibration, 22-5
simultaneous buffered waveform acquisition and generation, 14-2
AI Control VI, 9-7
AI Hardware Config VI, 20-4
AI Read One Scan VI, 6-7
AI Read VI
advantages and disadvantages of reading backlog, A-1
asynchronous continuous acquisition using DAQ Occurrences, 7-11 to 7-12
conditional retrieval example, 8-12
conditional retrieval input cluster, 8-11
controlling startup times (note), 7-7
forcing time limit for, 9-5, 9-8
multiple-waveform acquisition, 7-4
one-point calibration, 22-5
SCXI temperature measurement, 21-8
simple-buffered analog input with multiple starts, 7-7
simultaneous buffered waveform acquisition and generation, 14-2
software triggering, 8-11
AI Sample Channel VI, 6-1 to 6-2
AI Sample Channels VI, 6-3
AI Single Scan VI
basic non-buffered application, 6-4
hardware-timed analog I/O control loops, 6-8 to 6-9
improving control loop performance, 6-9 to 6-10
multiple-channel single-point analog input, 6-4
one-point calibration, 22-5
software-timed analog I/O control loops, 6-6
AI Start VI
amplifier offset, 21-5
hardware-timed analog I/O control loops, 6-8 to 6-9
multiple-waveform acquisition, 7-4
one-point calibration, 22-5
scan clock control, 9-7
SCXI temperature measurement, 21-8
simple-buffered analog input with multiple starts, 7-7
simultaneous buffered waveform acquisition and generation
hardware triggered, 14-3
software triggered, 14-2
Am9513 counter
continuous pulse train generation, 24-11 to 24-12
controlling pulse width measurement, 25-6 to 25-7
counting operations with no counters available, 24-21 to 24-22
description, 23-5
determining pulse width, 25-4 to 25-5
dividing frequencies, 28-2 to 28-3
events or elapsed time counting
connecting counters, 27-2 to 27-3
elapsed time, 27-9 to 27-10
events, 27-5 to 27-6

finite pulse train generation, 24-15 to 24-16
frequency and period measurement
connecting counters, 26-3
high frequency signals, 26-5, 26-6
how and when to measure, 26-2
low frequency signals, 26-9 to 26-10
internal timebases with maximum pulse width measurements (table), 25-9
single square pulse generation, 24-5 to 24-7
square pulse generation, 24-3
stopping counter generations, 24-24
amplification
increasing signal-to-noise ratio (figure), 18-4
methods for minimizing noise (note), 18-4
amplifier offset, reading, 21-5 to 21-6
AMUX-64T devices
addressing with MIO boards, A-1
analog input channel range (table), 5-13
channel addressing, 5-13 to 5-17
scanning order for DAQ devices, 5-14 to 5-17
four AMUX-64Ts (table), 5-16
one or two AMUX-64Ts (table), 5-15
specifying number for AMUX-64T device (table), 5-17
analog input. *See also* buffered waveform acquisition.
AMUX-64T external multiplexer device, 5-13 to 5-17
analog input/output control loops, 6-6 to 6-10
channel clock control, 9-3 to 9-6, 9-9
circular-buffered analog input examples, 7-13 to 7-14
continuous acquisition from multiple channels, 7-10 to 7-11
defining signals, 5-1 to 5-2
digital triggering, 8-2 to 8-5
external control of acquisition rate, 9-1 to 9-3
hardware triggering, 8-1 to 8-9
measurement systems, 5-4 to 5-6
multiple waveform acquisition, 7-3 to 7-5
multiple-channel single point analog input, 6-3 to 6-5
scan clock control, 9-6 to 9-8, 9-9
SCXI applications for measuring temperature (example), 21-2 to 21-13
selecting input settings, 5-7 to 5-12
calculating code width, 5-7
considerations for selecting, 5-7 to 5-8
differential measurement system, 5-9 to 5-10
measurement precision for various device ranges and limit settings (table), 5-8
nonreferenced single-ended measurement system, 5-12 to 5-13
referenced single-ended measurement system, 5-11
signals, 4-3, 5-1 to 5-6
simultaneous scan and channel clock control, 9-9
single waveform acquisition, 7-2 to 7-3
single-buffered analog input examples, 7-5 to 7-8
single-channel single point analog input, 6-1 to 6-2
software triggering, 8-9 to 8-12
terminology, 5-17
triggering, 8-6 to 8-9
Analog Input palette, 6-1
analog input SCXI modules
applications for measuring temperature (example), 21-2 to 21-13
multiplexed mode, 19-5 to 19-6
parallel mode, 19-6 to 19-7

analog input signals
choosing a measurement system, 5-4 to 5-6
choosing between analog and digital signals, 4-3
defining signals, 5-1 to 5-2
device range, 5-5
floating signal sources, 5-3
grounded signal sources, 5-2
referenced and non-referenced, 5-2
resolution of ADC, 5-4
signal limit settings, 5-6
types of analog signals (figure), 5-1
analog input/output control loops, 6-6 to 6-10
hardware-timed control loops, 6-7 to 6-9
improving performance, 6-9 to 6-10
overview, 6-6
software-timed control loops, 6-6 to 6-7
Analog IO Control Loop (HW timed) VI, 6-7
Analog IO Control Loop VI, 6-6 to 6-7
analog multiplexers (AMUX), 5-9. *See also* AMUX-64T devices.
analog output
buffered
overview, 10-1 to 10-2
stored in 2D arrays, 3-15 to 3-16
waveform generation, 12-1 to 12-3
circular buffered, 12-3 to 12-5
eliminating errors, 12-5
multiple-immediate updates, 11-3
SCXI analog output application example, 21-17
single-immediate updates, 11-1 to 11-2
single-point output
choosing between single-point or multiple-point generation, 4-4
overview, 10-1
analog output SCXI modules
application example, 21-17
multiplexed mode, 19-5
analog triggering
description, 8-6 to 8-7
diagram, 8-6
examples, 8-8 to 8-9
timeline for post-triggered data acquisition (figure), 8-7
analog-to-digital converter (ADC). *See* ADC.
anlog_io folder, for DAQ examples, 3-2
anlogin folder, for DAQ examples, 3-2
anlogout folder, for DAQ examples, 3-2
AO Clear VI
circular-buffered output, 12-5
simultaneous buffered waveform acquisition and generation, 14-2
waveform generation, 12-3
AO Config VI
analog output SCXI example, 21-17
circular-buffered output, 12-4
simultaneous buffered waveform acquisition and generation, 14-2
waveform generation, 12-3
AO Continuous Gen VI, 12-4
AO Generate Waveforms VI, 12-1
AO Group Config VI, 21-17
AO Hardware Config VI, 21-17
AO Single Update VI
analog output SCXI example, 21-17
calibrating SCXI modules for signal generation, 22-7
AO Start VI
circular-buffered output, 12-5
external control of update clock, 13-2
simultaneous buffered waveform acquisition and generation, 14-2
waveform generation, 12-3
AO Trigger and Gate Config VI, 14-3
AO Update Channel VI, 11-2
AO Update Channels VI, 11-1
AO Wait VI, 12-3
AO Waveform Gen VI, 12-2
AO Write One Update VI
analog I/O control loops, 6-7

multiple-immediate updates, 11-3
single-immediate updates, 11-2
AO Write VI
circular-buffered output, 12-5
simultaneous buffered waveform acquisition and generation, 14-2
waveform generation, 12-3
Array & Cluster option, 3-15
arrays
transposing, 3-15, 7-6
two-dimensional (2D) arrays, 3-14 to 3-16
AT bus devices, LabVIEW support for (table), 2-8

## B

bipolar range, 3-13, 5-7
breakpoints, setting, 29-4
buffered handshaking, 17-6 to 17-10
circular buffered examples, 17-9 to 17-10
simple buffered examples, 17-7 to 17-9
buffered pulse and period measurement, 25-7 to 25-8
buffered waveform acquisition, 7-1 to 7-14
circular-buffered analog input, 7-8 to 7-14
asynchronous continuous acquisition using DAQ occurrences, 7-11 to 7-12
continuous acquisition from multiple channels, 7-10 to 7-11
determining adequate buffer capacity, A-2
examples, 7-13 to 7-14
how circular buffers work, 7-9 to 7-10
overview, 7-8 to 7-10
simple-buffered analog input
displaying waveforms on graphs (example), 7-5 to 7-6
how buffers work, 7-2
multiple-waveform acquisition, 7-3 to 7-5
overview, 7-1
sampling with multiple starts (example), 7-6 to 7-7
single-waveform acquisition, 7-2 to 7-3
waiting to analyze data, 7-1 to 7-2
simultaneous acquisition and generation, 14-1 to 14-6
E-series MIO boards, 14-1 to 14-4
Lab/1200 boards, 14-6
legacy MIO boards, 14-4 to 14-6
buffered waveform generation
buffered analog output, 12-1 to 12-3
choosing between single-point or multiple-point generation, 4-4
circular-buffered output, 12-3 to 12-5
eliminating errors, 12-5
overview, 10-1 to 10-2
stored in 2D arrays, 3-15

## C

calibration. *See* SCXI calibration.
cascading counters
defined, 27-2
external connections (figure), 27-2, 27-3
channel addressing, 3-8 to 3-11
AMUX-64T devices, 5-13 to 5-17
analog input channel range (table), 5-13
scanning order, 5-14 to 5-17
channel name addressing, 3-9
channel number addressing, 3-10 to 3-11
SCXI modules, 20-1 to 20-2
channel clock, 9-3 to 9-6
channel and scan intervals using channel clock (figure), 9-1
considerations for specific devices (notes), 9-5

controlling externally, 9-3 to 9-6
rate parameter, 5-17
setting channel clock rate, 9-3
simultaneous control of scan and channel clocks, 9-9
TTL signal (example), 9-3
channel configuration, in NI-DAQ 5.*x* or 6.*x*, 2-21
Channel to Index VI (note), 8-11
circular buffered digital I/O examples, 17-9 to 17-10
circular-buffered analog input
asynchronous continuous acquisition using DAQ occurrences, 7-11 to 7-12
continuous acquisition from multiple channels, 7-10 to 7-11
examples
basic circular-buffered analog input, 7-13
Cont Acq to File (binary) VI, 7-14
Cont Acq to File (scaled) VI, 7-14
Cont Acq to Spreadsheet File VI, 7-14
Cont Acq&Chart (buffered) VI, 7-14
Cont Acq&Graph (buffered) VI, 7-14
how circular buffers work (figure), 7-9
overview, 7-8 to 7-10
circular-buffered analog output
changing waveform during generation, 12-3 to 12-5
eliminating errors, 12-5
clocks. *See* channel clock; scan clock; update clock.
code width, calculating, 5-7
cold junction compensation, 21-3 to 21-5
column major order, 3-14 to 3-15
common questions about LabVIEW data acquisition, A-1 to A-3
common-mode voltage
defined, 5-10
illustration, 5-10
conditional retrieval, 8-9. *See also* software triggering.
configuration. *See* installation and configuration.
Cont Acq to File (binary) VI, 7-14
Cont Acq to File (scaled) VI, 7-14, 12-6
Cont Acq to Spreadsheet File VI, 7-14
Cont Acq&Chart (Async Occurrence) VI, 7-11 to 7-12
Cont Acq&Chart (buffered) VI, 7-14
Cont Acq&Graph (buffered) VI, 7-14
Cont Acquire&Chart (immediate) VI, 6-5
Cont Pulse Train (8253) VI, 24-13 to 24-14, 28-3
Cont Pulse Train-Easy (9513) VI, 24-11
Cont Pulse Train-Easy (DAQ-STC) VI, 24-11
Cont Pulse Train-Int (9513) VI, 24-12
Cont Pulse Train-Int (DAQ-STC) VI, 24-12
continuous acquisition from multiple channels, 7-10 to 7-11
Continuous Generation example VI, 12-4
Continuous Pulse Generator Config VI
continuous pulse train generation, 24-12
finite pulse train generation, 24-16
continuous pulse train generation, 24-10 to 24-14
8253/54, 24-13 to 24-14
TIO-ASIC, DAQ-STC, and Am9513, 24-11 to 24-12
Continuous Transducer VI, 21-6
control loops. *See* analog input/output control loops.
Convert RTD Reading VI, 21-12 to 21-13
Convert Strain Gauge Reading VI, 21-15 to 21-16
Convert Thermocouple Reading VI, 21-8
Count Events (8253) VI, 27-7
Count Events or Time Easy VI
events, 27-3
time
Am9513, 27-10

TIO-ASIC and DAQ-STC, 27-8
Count Events-Easy (9513) VI, 27-5
Count Events-Easy (DAQ-STC) VI, 27-3
Count Events-Int (9513) VI, 27-5 to 27-6
Count Events-Int (DAQ-STC) VI, 27-4
Count Time (8253) VI, 27-12
Count Time-Easy (9513) VI, 27-10
Count Time-Easy (DAQ-STC) VI, 27-8
Count Time-Int (9513) VI, 27-11
Count Time-Int (DAQ-STC) VI, 27-9
counter addressing for VIs, 3-8 to 3-11
counter chips used in National Instruments devices, 23-4 to 23-6. *See also* 8253/54 counter; AM9513 counter; DAQ-STC counter.
counter folder, for DAQ examples, 3-2
Counter Read VI
controlling pulse width measurement, 25-6
counting events
Am9513, 27-6
DAQ-STC, 27-4
counting time
Am9513, 27-11
DAQ-STC, 27-9
measuring frequency and period
high frequency signals, 26-6
low frequency signals, 26-10
Counter Start VI
continuous pulse train generation, 24-12
controlling pulse width measurement, 25-6
counting events
Am9513, 27-6
DAQ-STC, 27-4
counting time
Am9513, 27-11
DAQ-STC, 27-9
dividing frequencies, 28-2
finite pulse train generation, 24-16
measuring frequency and period
high frequency signals, 26-6
low frequency signals, 26-10
single square pulse generation, 24-6
Counter Stop VI
controlling pulse width measurement, 25-6
counting events
Am9513, 27-6
DAQ-STC, 27-4
counting time
Am9513, 27-11
DAQ-STC, 27-9
dividing frequencies, 28-2
finite pulse train generation, 24-15
measuring frequency and period
high frequency signals, 26-6
low frequency signals, 26-10
stopping counter generations, 24-24
counters
accuracy of counters, 24-23
basic functions, 23-1 to 23-6
capabilities, 23-1
choosing between counting methods, 4-5
counting events or elapsed time, 27-1 to 27-12
connecting counters, 27-1 to 27-3
elapsed time, 27-8 to 27-12
events, 27-3 to 27-7
counting operations with no counters available, 24-21 to 24-22
digital *vs.* counter interfacing, 4-4
dividing frequencies, 28-1 to 28-3
frequency and period measurement, 26-1 to 26-10
connecting counters for measuring, 26-3
high frequency signals, 26-4 to 26-8
how and when to measure, 26-1 to 26-2
low frequency signals, 26-8 to 26-10
gating modes (figure), 23-4

pulse train generation, 24-10 to 24-21
continuous pulse train, 24-10 to 24-14
finite pulse train, 24-14 to 24-21
pulse width measurement, 25-1 to 25-9
controlling pulse width measurement, 25-6 to 25-7
determining pulse width, 25-2 to 25-6
increasing measurable width range, 25-9
square pulse generation, 24-1 to 24-10
8253/54, 24-4
single square pulse generation, 24-4 to 24-10
TIO-ASIC, DAQ-STC, and Am9513, 24-3
stopping counter generations, 24-24
timebase uncertainty, 24-23
CTR Buffer Config VI, 25-8
CTR Buffer Read VI, 25-8
CTR Control VI
buffered pulse and period measurement, 25-8
enabling and disabling FOUT signal, 24-21
measuring frequency and period, 26-6
CTR Group Config VI, 25-8
CTR Mode Config VI
buffered pulse and period measurement, 25-8
finite pulse train generation, 24-17
current setting for VIs, 3-6
current value conventions for VIs, 3-6

## D

daisy chaining SCXI chassis, 21-21 to 21-22
DAQ Channel Wizard
channel name addressing, 3-9
limit settings, 3-11
SCXI programming considerations (note), 20-1
DAQ devices *vs.* SCXI devices, 4-3
DAQ examples
location of example files, 3-2
locations, 3-1 to 3-2
DAQ hardware. *See* hardware; installation and configuration.
DAQ Occurrence Config VI, 7-11 to 7-12
DAQ Solution Wizard, 3-1
DAQ VIs. *See* VIs.
DAQ-STC counter
continuous pulse train generation, 24-11 to 24-12
controlling pulse width measurement, 25-6 to 25-7
counting operations with no counters available, 24-21 to 24-22
description, 23-5
determining pulse width, 25-2 to 25-3
dividing frequencies, 28-2 to 28-3
events or elapsed time counting
events, 27-3 to 27-4
time, 27-8 to 27-12
finite pulse train generation
using Advanced VIs, 24-17
using Easy and Intermediate VIs, 24-15 to 24-16
frequency and period measurement
connecting counters, 26-3
high frequency signals, 26-4, 26-6
how and when to measure, 26-2
low frequency signals, 26-8 to 26-9, 26-10
internal timebases with maximum pulse width measurements (table), 25-9
single square pulse generation, 24-5 to 24-7
square pulse generation, 24-3
stopping counter generations, 24-24
data acquisition. *See also* analog input; VIs.

analog input/output control loops, 6-6 to 6-10
basic LabVIEW data acquisition concepts, 3-1 to 3-16
data organization for analog applications, 3-14 to 3-16
limit settings, 3-11 to 3-13
location of common DAQ examples, 3-1 to 3-2
buffered. *See* buffered waveform acquisition.
common questions about LabVIEW data acquisition, A-1 to A-3
important terms, 5-17
multiple-channel single-point, 6-3 to 6-5
single-channel single-point, 6-1 to 6-2
triggered. *See* triggered data acquisition.
data acquisition hardware. *See* hardware.
Data Acquisition palette, 3-4
data organization for analog applications, 3-14 to 3-16
column major order, 3-14 to 3-15
row major order, 3-14
two-dimensional (2D) arrays, 3-14 to 3-16
data types for LabVIEW, *xxiii*
debugging VIs, 29-1 to 29-4
error handling, 29-2 to 29-3
execution highlighting, 29-3
hardware connection errors, 29-1
setting breakpoints and showing advanced DAQ VIs, 29-4
single-stepping through VIs, 29-3
software configuration errors, 29-1 to 29-2
using Probe tool, 29-4
VI construction errors, 29-2 to 29-4
default input for VIs, 3-6
default setting for VIs, 3-6
Delayed Pulse (8253) VI, 24-7 to 24-10
Delayed Pulse Generator Config VI
finite pulse train generation, 24-16
measuring frequency and period, 26-6
single square pulse generation, 24-6
Delayed Pulse-Easy (9513) VI, 24-5
Delayed Pulse-Easy (DAQ-STC) VI, 24-5
Delayed Pulse-Int (9513) VI, 24-6
Delayed Pulse-Int (DAQ-STC) VI, 24-6
delays for improving control loop performance, 6-9 to 6-10
device range
considerations for selecting analog input settings, 5-7 to 5-8
description, 5-5
effect on ADC precision (figure), 5-5
measurement precision for various ranges and limit settings (table), 5-8
setting range and polarity, 3-13
diagnostic resources, online, B-1
differential measurement system, 5-9 to 5-10
8-channel differential system (figure), 5-9
common mode voltage (figure), 5-10
when to use, 5-10
digital and relay SCXI modules, 19-6
Digital Clock Config VI, 17-8
digital folder, for DAQ examples, 3-2
digital I/O
buffered handshaking, 17-6 to 17-10
circular buffered examples, 17-9 to 17-10
simple buffered examples, 17-7 to 17-9
choosing between non-latched or latched digital I/O, 4-5
digital *vs.* counter interfacing, 4-4
handshaking (latched) digital I/O, 17-1 to 17-2
immediate (non-latched) digital I/O, 16-1 to 16-3
non-buffered handshaking, 17-5 to 17-6
overview, 15-1
SCXI application examples

digital input, 21-18 to 21-19
digital output, 21-19 to 21-20
sending out multiple digital values, 17-3 to 17-5
Digital Mode Config VI, 17-8
digital ports and lines, 15-1
digital SCXI application examples
digital input, 21-18 to 21-19
digital output, 21-19 to 21-20
digital SCXI modules
multiplexed mode for digital and relay modules, 19-6
parallel mode, 19-7
digital signals *vs.* analog signals, 4-3
digital triggering
defined, 8-2
description, 8-2 to 8-3
diagram of signal connections, 8-2
examples, 8-4 to 8-5
timeline for post-triggered data acquisition (figure), 8-3
DIO Buffer Control VI, 17-8 to 17-9
DIO Clear VI, 17-7
DIO Config VI, 17-8 to 17-9
DIO Group Config VI, 17-5
DIO Port Config VI
digital input application example, 21-18 to 21-19
immediate digital I/O, 16-3
DIO Single Read/Write VI, 17-5
DIO Start VI, 17-8
DIO Wait VI, 17-7
Disable Indexing option, 3-15
Display and Output Acq'd File (scaled) VI, 12-6
dividing frequencies, 28-1 to 28-3
documentation
about the manual, *xxi*
conventions used in manual, *xxi-xxiii*
flowchart for finding information, 4-2
how to use this book, 1-1 to 1-4
related documentation, *xxiv*
down counter, 28-1
Down Counter or Divide Config VI, 28-2

## E

E-series MIO boards
hardware triggered, 14-3 to 14-4
simultaneous buffered waveform acquisition and generation, 14-1 to 14-4
software triggered, 14-2 to 14-3
Easy Counter VI
continuous pulse train generation, 24-11
finite pulse train generation, 24-15
single square pulse generation, 24-6
Easy VIs. *See also* VIs.
addressing OUT and IN pins on DIO-32F board, A-2
continuous pulse train generation, 24-11
counting elapsed time
Am9513, 27-10
DAQ-STC, 27-8
counting events, 27-3 to 27-7
digital input application, 21-18
digital output application, 21-19
finite pulse train generation, 24-15
grouping two or more ports, A-2
immediate digital I/O, 16-1 to 16-2
limitations, 6-3
measuring frequency and period
high frequency signals, 26-4 to 26-5
low frequency signals, 26-8 to 26-9
multiple-channel single-point analog input, 6-3
multiple-immediate updates, 11-3
multiple-waveform acquisition, 7-3
overview, 3-4 to 3-5
single square pulse generation, 24-6
single-channel single-point analog input, 6-1
single-immediate updates, 11-1 to 11-2

single-waveform acquisition, 7-2 to 7-3
strain gauge application, 21-15
waveform generation, 12-1 to 12-2
edges of signals, 23-2
EEPROM, for storing calibration constants, 22-1 to 22-2
default load area, 22-2
factory area, 22-2
user area, 22-2
elapsed time counting. *See* events or elapsed time counting.
Error Handler VIs, 29-2
error handling
debugging VIs, 29-2 to 29-3
error in and error out output clusters, 3-8
Event or Time Counter Config VI
counting events
Am9513, 27-6
DAQ-STC, 27-4
counting time
Am9513, 27-11
DAQ-STC, 27-9
measuring frequency and period, 26-6
events or elapsed time counting, 27-1 to 27-12
adjacent counters for counter chips (table), 27-2
connecting counters, 27-1 to 27-3
Am9513, 27-2 to 27-3
cascading counters (figure), 27-2
external connections (figures), 27-1
elapsed time, 27-8 to 27-12
8253/54, 27-12
Am9513, 27-9 to 27-10
TIO-ASIC and DAQ-STC, 27-8 to 27-9
events, 27-3 to 27-7
8253/54, 27-7
Am9513, 27-5 to 27-6
TIO-ASIC and DAQ-STC, 27-3 to 27-4
execution highlighting, 29-3
external control
acquisition rate, 9-1 to 9-9
channel and scan intervals using channel clock (figure), 9-1
channel clock control, 9-3 to 9-6
choosing between triggering and external clock control, 4-4
description, 9-1 to 9-3
round-robin scanning (figure), 9-2
scan clock control, 9-6 to 9-8
simultaneous control of scan and channel clocks, 9-9
update clock, 13-1 to 13-3
Generate N Updates-ExtUpdateClk VI, 13-1 to 13-2
input pins (table), 13-2
supplying test clock from DAQ device, 13-3
external conversions, 9-4 to 9-5
external devices, LabVIEW support for (table), 2-7
EXTUPDATE* signal (table), 13-2

## F

filtering, 18-5
Finite Pulse Train (8253) VI, 24-18 to 24-21
finite pulse train generation, 24-14 to 24-21
8253/54, 24-18 to 24-21
DAQ-STC, 24-17 to 24-18
physical connections (figure), 24-14
TIO-ASIC, DAQ-STC, and Am9513, 24-15 to 24-16
Finite Pulse Train-Adv (DAQ-STC) VI, 24-17
Finite Pulse Train-Easy (9513) VI, 24-15
Finite Pulse Train-Easy (DAQ-STC) VI, 24-15
Finite Pulse Train-Int (9513) VI, 24-15
Finite Pulse Train-Int (DAQ-STC) VI, 24-15
floating signal sources, 5-3
FOUT pin, 13-3, 24-21

FREQ_OUT pin, 13-3, 24-21
frequency and period measurement, 26-1 to 26-10
connecting counters for measuring, 26-3
equation for obtaining measurements, 26-2
high frequency signals, 26-4 to 26-8
how and when to measure, 26-1 to 26-2
low frequency signals, 26-8 to 26-10
square wave frequency measurement (figure), 26-1
square wave period measurement (figure), 26-2
frequency division, 28-1 to 28-3
8253/54, 28-3
TIO-ASIC, DAQ-STC, and Am9513, 28-2 to 28-3
wiring (figure), 28-1
Function Generator VI, 12-5
Functions palette
Array & Cluster, 3-15
DAQ, 6-1
illustration, 3-3
locating VIs, 3-3

## G

gain, defined, 3-13
gains (SCXI)
default gain, 20-3
description, 20-3 to 20-4
SCXI-1100 channel arrays, input limits array, and gains (table), 20-4
GATE input, for counters, 23-2
General Error Handler VI
debugging VIs, 29-2
pulse width measurement, 25-6
Generate Continuous Sinewave VI, 12-3
Generate Delayed Pulse VI
single square pulse generation, 24-5
stopping counter generations, 24-24
Generate N Updates example VI, 12-2
Generate N Updates-ExtUpdateClk VI, 13-1 to 13-2
Generate Pulse Train on FOUT VI, 13-3, 24-22
Generate Pulse Train on FREQ_OUT VI, 13-3, 24-22
Generate Pulse Train VI
continuous pulse train generation
8253/54, 24-13 to 24-14
TIO-ASIC, DAQ-STC, and Am9513, 24-11
finite pulse train generation, DAQ-STC and Am9513, 24-15
stopping counter generations, 24-24
supplying external test clock, 13-3
Get DAQ Device Information VI, 2-1
Get Timebase (8253) VI, 25-5, 26-7
Getting Started Analog Input example VI
channel clock control (figure), 9-4
reading amplifier offset, 21-5
scan clock control (figure), 9-8
temperature sensor, 21-4
graphing simple-buffered analog input (example), 7-5 to 7-6
grounded signal sources, 5-2

## H

handshaking (latched) digital I/O, 17-1 to 17-10
buffered handshaking, 17-6 to 17-10
circular buffered examples, 17-9 to 17-10
simple buffered examples, 17-7 to 17-9
connecting signal lines
digital input (figure), 17-3
digital output (figure), 17-4
DAQ devices supporting digital handshaking, 17-1

defined, 15-2
grouping ports for DIO-32 devices (note), 17-4
non-buffered handshaking, 17-5 to 17-6
overview, 17-1 to 17-2
sending out multiple digital values, 17-3 to 17-5
hardware. *See also* installation and configuration.
debugging connection errors, 29-1
LabVIEW data acquisition hardware support (table), 2-4 to 2-13
relationship between LabVIEW, NI-DAQ, and DAQ hardware (figure), 2-3
hardware triggering, 8-1 to 8-9
analog
description, 8-6 to 8-7
examples, 8-8 to 8-9
digital
description, 8-2 to 8-3
examples, 8-4 to 8-5
overview, 8-1
hardware-timed analog input/output control loops, 6-7 to 6-9

## I

IBF (Input Buffer Full) line, 17-2
ICTR Control VI
counting events, 27-7
determining pulse width, 25-5 to 25-6
measuring frequency and period, 26-7
stopping counter generations, 24-24
ICTR Control-Int VI, 27-12
ICTR Timebase Generator VI, 24-19 to 24-20
immediate digital I/O. *See* nonlatched digital I/O.
immediate updates
multiple, 11-3
single, 11-1 to 11-2
Index Array function, 3-15
initialization of data acquisition devices, A-3
Input Buffer Full (IBF) line, 17-2
input range, and input setting selection, 5-7 to 5-8
installation and configuration
channel configuration in NI-DAQ 5.*x* or 6.*x*, 2-21
DAQ devices
installing and configuring (figure), 2-2
using NI-DAQ 4.8.*x* on Macintosh, 2-14 to 2-16
using NI-DAQ 5.*x* or 6.*x*, 2-14
debugging software configuration errors, 29-1 to 29-2
LabVIEW data acquisition hardware support (table), 2-4 to 2-13
relationship between LabVIEW, NI-DAQ, and DAQ hardware (figure), 2-3
SCXI chassis
hardware configuration, 2-16 to 2-17
software configuration, 2-18 to 2-20
Intermediate VIs. *See also* VIs.
advantages, 6-5
asynchronous continuous acquisition using DAQ occurrences, 7-11 to 7-12
circular-buffered output, 12-4 to 12-5
continuous acquisition from multiple channels, 7-10 to 7-11
continuous pulse train generation, 24-12
controlling pulse width measurement, 25-6 to 25-7
counting elapsed time
8253/54, 27-12
Am9513, 27-11
DAQ-STC, 27-9
counting events
8253/54, 27-7
Am9513, 27-5 to 27-6

DAQ-STC, 27-4
dividing frequencies, 28-2 to 28-3
finite pulse train generation, 24-15 to 24-16
measuring frequency and period
high frequency signals, 26-6
low frequency signals, 26-10
multiple-channel single-point analog input, 6-4 to 6-5
multiple-waveform acquisition, 7-4 to 7-5
non-buffered handshaking, 17-5
overview, 3-5
SCXI temperature measurement examples, 21-6 to 21-8
simple buffered handshaking, 17-7
simultaneous buffered waveform acquisition and generation, 14-2
single square pulse generation, 24-6
single-immediate updates, 11-2
stopping counter generations, 24-24
strain gauge application, 21-15
waveform generation, 12-3
interval scanning, 5-17
isolation of transducer signals, 18-5

## L

Lab Series devices, LabVIEW support for (table), 2-9
Lab/1200 boards, simultaneous buffered waveform acquisition and generation, 14-6
LabVIEW software
basic LabVIEW data acquisition concepts, 3-1 to 3-16. *See also* VIs.
data organization for analog applications, 3-14 to 3-16
location of common DAQ examples, 3-1 to 3-2
common questions about LabVIEW, A-1 to A-3
data types, *xxiii*
relationship between LabVIEW, NI-DAQ, and DAQ hardware (figure), 2-3
latched digital I/O. *See* handshaking (latched) digital I/O.
legacy MIO boards
hardware triggered, 14-5 to 14-6
simultaneous buffered waveform acquisition and generation, 14-4 to 14-6
software triggered, 14-4 to 14-5
limit settings
considerations for selecting analog input settings, 5-7 to 5-8
description, 5-6
effect on ADC precision (figure), 5-6
measurement precision for various device ranges and limit settings (table), 5-8
SCXI gains, 20-3 to 20-4
VI limit settings, 3-11 to 3-13
linearizing voltage levels, 18-6

## M

Macintosh systems
configuring DAQ devices, 2-14 to 2-16
LabVIEW data acquisition hardware support (table), 2-4 to 2-13
NI-DAQ driver files, 2-3
SCXI chassis
hardware configuration, 2-16 to 2-17
software configuration, 2-18 to 2-20
manual. *See* documentation.
maximum sampling rate per channel, 7-5
Meas Buffered Pulse-Period (DAQ-STC) VI, 25-7
Measure Frequency VI, 26-4, 26-5
Measure Frequency < 1kHz (8253) VI, 26-8, 26-10
Measure Frequency > 1kHz (8253) VI, 26-7
Measure Frequency - Dig Start > 1kHz (8253) VI, 26-8

Measure Frequency-Easy (9513) VI, 26-5
Measure Frequency-Easy (DAQ-STC) VI, 26-4
Measure Period-Easy (9513) VI, 26-8, 26-9
Measure Period-Easy (DAQ-STC) VI, 26-8
Measure Pulse -Easy (9513) VI, 25-4
Measure Pulse-Easy (DAQ-STC) VI, 25-2
Measure Pulse Width or Period VI
  determining pulse width
    Am9513, 25-4 to 25-5
    DAQ-STC, 25-2 to 25-3
  measuring low frequency signals, 26-9
Measure Short Pulse Width (8253) VI, 25-5
measurement system
  choosing, 5-4 to 5-6
  differential measurement system, 5-9 to 5-10
  nonreferenced single-ended measurement system, 5-12 to 5-13
  referenced single-ended measurement system, 5-11
Microsoft Windows. *See* Windows environment.
MIO boards
  E-series MIO boards, 14-1 to 14-4
  legacy MIO boards, 14-4 to 14-6
multiple-channel single-point analog input, 6-3 to 6-5
multiple-immediate updates, 11-3
multiple-waveform acquisition
  choosing between single-point and multi-point acquisition, 4-4
  procedure for acquiring, 7-3 to 7-5
multiplexed mode (SCXI)
  analog input modules, 19-5
  analog output modules, 19-6
  channel addressing, 20-1 to 20-2
  digital and relay modules, 19-6
  SCXI-1200 (Windows), 19-6
My Single Scan Processing VI, 6-5

## N

National Instruments Web support, B-1 to B-2
NEC devices, LabVIEW support for (table), 2-9
NI-DAQ driver for Macintosh, 2-3
NI-DAQ software
  channel configuration in NI-DAQ 5.*x*, 6.*x*, 2-21
  configuring for SCXI modules
    NI-DAQ 4.8.*x* for Macintosh, 2-18 to 2-20
    NI-DAQ 5.*x*, 6.*x* for Windows, 2-18
  driver files
    Macintosh versions, 2-3
    versions of NI-DAQ drivers (note), 2-1
    Windows versions, 2-3
  installing
    NI-DAQ 4.8.*x* on Macintosh, 2-14 to 2-16
    NI-DAQ 5.*x* or 6.*x*, 2-14
  relationship between LabVIEW, NI-DAQ, and DAQ hardware (figure), 2-3
NIDAQ32.DLL file, 2-3
non-buffered handshaking, 17-5 to 17-6
nonlatched digital I/O
  defined, 15-2
  overview, 16-1 to 16-3
  using Easy Digital VIs, 16-1 to 16-2
non-referenced signal sources, 5-2
nonreferenced single-ended (NRSE) measurement system, 5-12 to 5-13
  16-channel NRSE system (figure), 5-12
  when to use, 5-12
NuBus devices, LabVIEW support for (table), 2-10
Nyquist frequency, 5-2
Nyquist Theorem, 5-2

## O

OBF (Output Buffer Full) line, 17-2
one-point calibration, 22-5 to 22-6
online problem-solving and diagnostic resources, B-1
OUT output pin, 23-2
OUT2 signal (table), 13-2
Output Buffer Full (OBF) line, 17-2

## P

parallel mode (SCXI)
  analog input modules, 19-6 to 19-7
  channel addressing, 20-1 to 20-2
  digital modules, 19-7
  SCXI-1200 (Windows), 19-7
parameters for VIs
  common DAQ VI parameters, 3-7
  conventions, 3-5 to 3-6
pattern generation, 17-2
PC Card devices, LabVIEW support for (table), 2-7
PCI bus devices, LabVIEW support for (table), 2-4 to 2-5
PC/XT bus devices, LabVIEW support for (table), 2-9
Period Meas Config VI, 25-6 to 25-7
period measurement. *See* frequency and period measurement.
PFI5/UPDATE* signal (table), 13-2
polling for analog input, 6-9 to 6-10
ports
  digital ports and lines, 15-1
  grouping ports without handshaking, A-2
  port addressing, 3-8 to 3-11
  writing to digital port while reading digital data, A-2
pressure measurement with strain gauges (example), 21-13 to 21-16
Probe tool, 29-4
problem-solving and diagnostic resources, online, B-1
pulse generation, square. *See* square pulse generation.
pulse train generation, 24-10 to 24-21
  8253/54, 24-4
  continuous pulse train, 24-10 to 24-14
    8253/54, 24-13 to 24-14
    TIO-ASIC, DAQ-STC, and Am9513, 24-11 to 24-12
  duty cycles (figure), 24-2
  finite pulse train, 24-14 to 24-21
    8253/54, 24-18 to 24-21
    DAQ-STC, 24-17 to 24-18
    physical connections (figure), 24-14
    TIO-ASIC, DAQ-STC, and Am9513, 24-15 to 24-16
  TIO-ASIC, DAQ-STC, and Am9513, 24-3
pulse width measurement, 25-1 to 25-9
  buffered pulse and period measurement, 25-7 to 25-8
  controlling pulse width measurement, 25-6 to 25-7
  counting input signals (figure), 25-1
  determining pulse width, 25-2 to 25-6
  increasing measurable width range, 25-9
  measuring pulse width, 25-1 to 25-2
  overview, 25-1
  physical connections for determining pulse width (figure), 25-2
Pulse Width VI, 25-6 to 25-7
Pulse Width or Period Meas Config VI, 26-10
pulsed counter signal generation, 24-1
PXI bus devices, LabVIEW support for (table), 2-6
PXI Combo chassis, LabVIEW support for (table), 2-5

## Q

questions
about using DAQ devices, 4-3 to 4-5
LabVIEW data acquisition questions, A-1 to A-3

## R

range. *See* device range.
Read from Digital Line VI, 16-2
Read from Digital Port VI
digital input application, 21-19
immediate digital I/O, 16-2
referenced signal sources, 5-2
referenced single-ended (RSE) measurement system, 5-11
relay SCXI modules, 19-6
Remote SCXI, sampling rate limits (note), 19-4
REQ (Request) line, 17-2
Resistance-Temperature Detectors (RTDs), 21-10 to 21-13
resolution of ADC, 5-4
round-robin scanning (figure), 9-2
row major order, 3-14
RSE (referenced single-ended) measurement system, 5-11
RTD Conversion VI, 21-12
RTDs for measuring temperature, 21-10 to 21-13
run_me.llb library of DAQ examples, 3-2

## S

SC-2042 RTD device, 21-11
Scale Constant Tuner VIs, 22-7
Scaling Constant Tuner VI, 21-6, 21-8
scan clock, 9-6 to 9-8
channel and scan intervals using channel clock (figure), 9-1
devices without scan clocks (note), 9-7
input pins (table), 9-7
MIO devices without scan clocks (note), 9-7
scan-clock orientation of LabVIEW, 9-2
simultaneous control of scan and channel clocks, 9-9
scans
channel clock rate parameter, 5-17
defined, 5-17
interval scanning, 5-17
maximum scan rate, calculating, 7-5
number of samples parameter, 5-17
number of scans to acquire parameter, 5-17
round-robin scanning (figure), 9-2
scan rate parameter, 5-17
SCXI 1124 Update Channels VI, 21-17
SCXI application examples, 21-1 to 21-22
analog input application for measuring temperature, 21-2 to 21-13
analog output application, 21-17
DAQ example files, 3-2
digital input application, 21-18 to 21-19
digital output application, 21-19 to 21-20
multi-chassis applications, 21-21 to 21-22
overview, 21-1
pressure measurement with strain gauges, 21-13 to 21-16
temperature measurement applications
amplifier offset, 21-5 to 21-6
sensors for cold-junction compensation, 21-3 to 21-5
using RTDs, 21-10 to 21-13
using thermocouples, 21-2 to 21-3
VI examples, 21-6 to 21-10
SCXI Cal Constants VI
calculation of calibration constants, 22-2
calibrating SCXI modules for signal generation, 22-8
loading saved calibration constants, 22-7, 22-8

one-point calibration, 22-6
overwriting default constants in EEPROM (note), 22-2
two-point calibration, 22-6
SCXI calibration, 22-1 to 22-8
EEPROM for storing calibration constants, 22-1 to 22-2
default load area, 22-2
factory area, 22-2
user area, 22-2
one-point calibration, 22-5 to 22-6
overview, 22-3
signal acquisition, 22-4 to 22-7
signal generation, 22-7 to 22-8
two-point calibration, 22-6 to 22-7
SCXI chassis
illustration, 19-4
LabVIEW support for (table), 2-10
SCXI modules
components
chassis (figure), 19-4
illustration, 19-3
overview, 19-2 to 19-3
hardware configurations
illustration, 19-2
overview, 19-1
procedure, 2-16 to 2-17
LabVIEW support for (table), 2-11
sampling rate limits for Remote SCXI (note), 19-4
software configuration
Macintosh systems, 2-18 to 2-20
Windows environment, 2-18
when to use, 4-3
SCXI operating modes, 19-5 to 19-7
multiplexed mode
analog input modules, 19-5
analog output modules, 19-6
channel addressing, 20-1 to 20-2
digital and relay modules, 19-6
SCXI-1200 (Windows), 19-6
parallel mode
analog input modules, 19-6 to 19-7
channel addressing, 20-1 to 20-2
digital modules, 19-7
SCXI-1200 (Windows), 19-7
SCXI programming considerations, 20-1 to 20-5
channel addressing, 20-1 to 20-2
gains, 20-3 to 20-5
SCXI-1100 channel arrays, input limits array, and gains (table), 20-4
settling time, 20-5
SCXI Temperature Monitor VI, 21-9
SCXI thermocouple example VIs
overview, 21-4 to 21-5
SCXI-1100 Thermocouple VI, 21-7
SCXI-1120/1121 Thermocouple example VI, 21-10
SCXI-1100 Thermocouple VI, 21-7
SCXI-1100 Voltage example, 21-6
SCXI-1120/1121 Thermocouple example VI, 21-10
SCXI-116x Digital Output VI, 21-20
SCXI-1162/1162HV Digital Input VI, 21-19
SCXI-1200 module
multiplexed mode (Windows), 19-6
parallel mode (Windows), 19-7
settling time (SCXI), 20-5
Show Help option, 3-2
Show VI Info option, 3-2
signal conditioning
amplification, 18-4
common transducers (table), 18-1 to 18-2
common types of signal conditioning, 18-2
conditioning for common types of transducers/signals (figure), 18-3
defined, 18-2
filtering, 18-5
isolation, 18-5
linearization, 18-6

transducer excitation, 18-5
Signal Conditioning Components (SCC), LabVIEW support for (table), 2-12 to 2-13
signal divider, 28-1
signal edges, 23-2
signal voltage range. *See* limit settings.
signals. *See also* analog input signals.
choosing between analog and digital signals, 4-3
simple-buffered analog input
examples
displaying waveforms on graphs, 7-5 to 7-6
sampling with multiple starts, 7-6 to 7-7
multiple-waveform acquisition, 7-3 to 7-5
overview, 7-1
single-waveform acquisition, 7-2 to 7-3
waiting to analyze data, 7-1 to 7-2
Simple Error Handler VI
analog output SCXI example, 21-17
debugging VIs, 29-2 to 29-3
multiple-channel single-point analog input, 6-5
single-immediate updates, 11-2
Simul AI/AO Buffered (E-series MIO) VI, 14-2 to 14-3
Simul AI/AO Buffered (Lab/1200) VI, 14-6
Simul AI/AO Buffered (legacy MIO) VI, 14-4 to 14-5
Simul AI/AO Buffered Trigger (E-series MIO) VI, 14-3 to 14-4
Simul AI/AO Buffered Trigger (Lab/1200) VI, 14-6
Simul AI/AO Buffered Trigger (legacy MIO) VI, 14-5
simultaneous buffered waveform acquisition and generation. *See* buffered waveform acquisition.
single-channel single-point analog input
choosing between single-point and multi-point acquisition, 4-4
description, 6-1 to 6-2
single-ended measurement system
nonreferenced, 5-12 to 5-13
referenced, 5-11
single-immediate updates, 11-1 to 11-2
single-point analog output
choosing between single-point or multiple-point generation, 4-4
overview, 10-1
single-stepping through VIs, 29-3
single-waveform acquisition, 7-2 to 7-3
software configuration errors, debugging, 29-1 to 29-2
software-related resources, B-2
software-timed analog input/output control loops, 6-6 to 6-7
software timing, 10-1
software triggering
conditional retrieval examples, 8-12
description, 8-9 to 8-12
timeline of conditional retrieval (figure), 8-10
solution folder, for DAQ examples, 3-2
SOURCE input, for counters, 23-2
spreadsheet files
Cont Acq to Spreadsheet File VI, 7-14
simple buffered-analog input example, 7-8
square pulse generation, 24-1 to 24-10
8253/54, 24-4
duty cycle (figure), 24-2
overview, 24-1 to 24-2
single square pulse generation, 24-4 to 24-10
8253/54, 24-7 to 24-10
TIO-ASIC, DAQ-STC, and Am9513, 24-5 to 24-7
terminology related to, 24-1

TIO-ASIC, DAQ-STC, and Am9513, 24-3
square wave frequency, measuring (figure), 26-1
STB (Strobe Input) line, 17-2
Strain Gauge Conversion VI, 21-15
strain gauges for measuring pressure (example), 21-13 to 21-16
Strobe Input (STB) line, 17-2

## T

technical support resources, B-1 to B-2
temperature measurement applications (SCXI)
amplifier offset, 21-5 to 21-6
sensors for cold-junction compensation, 21-3 to 21-5
using RTDs, 21-10 to 21-13
using thermocouples, 21-2 to 21-3
VI examples, 21-6 to 21-10
terminal count (TC), 23-3
Thermistor Conversion VI, 21-4
thermocouples
measuring temperature (example), 21-2 to 21-3
SCXI Thermocouple example VIs, 21-4
SCXI-1100 Thermocouple VI, 21-6
SCXI-1120/1121 Thermocouple example VI, 21-10
timebase period uncertainty, 24-23
TIO-ASIC counter
continuous pulse train generation, 24-11 to 24-12
controlling pulse width measurement, 25-6 to 25-7
counting operations with no counters available, 24-21 to 24-22
description, 23-5
determining pulse width, 25-2 to 25-3
dividing frequencies, 28-2 to 28-3
event counting, 27-3 to 27-4
finite pulse train generation, 24-15 to 24-16
frequency and period measurement
high frequency signals, 26-4, 26-6
how and when to measure, 26-2
low frequency signals, 26-8 to 26-9
internal timebases with maximum pulse width measurements (table), 25-9
single square pulse generation, 24-5 to 24-7
square pulse generation, 24-3
toggled counter signal generation, 24-1
transducers
common transducers (table), 18-1 to 18-2
excitation, 18-5
linearization, 18-6
signal conditioning for common types of transducers/signals (figure), 18-3
Transpose 2D Array option (note), 3-15
transposing arrays, 3-15, 7-6
triggered data acquisition, 8-1 to 8-12
analog hardware triggering
description, 8-6 to 8-7
examples, 8-8 to 8-9
deciding which digital trigger setting to use, A-3
digital hardware triggering
description, 8-2 to 8-3
examples, 8-4 to 8-5
hardware triggering, 8-1 to 8-9
overview, 8-1
software triggering
conditional retrieval examples, 8-12
description, 8-9 to 8-12
triggering *vs.* external clock control, 4-4
triggering, defined, 8-1
triggers, defined, 8-1
two dimensional (2D) arrays, 3-14 to 3-16
analog output buffers, 3-15 to 3-16
column major order, 3-14 to 3-15
extracting single channel, 3-15

illustration, 3-14
row major order, 3-14
two-point calibration, 22-6 to 22-7

## U

unipolar range, 3-13, 5-7
update clock, controlling externally, 13-1 to 13-3
Generate N Updates-ExtUpdateClk VI, 13-1 to 13-2
input pins (table), 13-2
overview, 13-1
supplying test clock from DAQ device, 13-3
Utility VIs, 3-5

## V

VIs. *See also* Advanced VIs; Easy VIs; Intermediate VIs.
channel, port, and counter addressing, 3-8 to 3-11
common DAQ VI parameters, 3-7
data organization for analog applications, 3-14 to 3-16
debugging, 29-1 to 29-4
default and current value conventions, 3-6
error handling, 3-8
finding VIs in LabVIEW, 3-3
limit settings, 3-11 to 3-13
organization, 3-4 to 3-5
parameter conventions, 3-5 to 3-6
SCXI examples, 21-6 to 21-10
Utility VIs, 3-5
VXI carriers and modules, LabVIEW support for (table), 2-10

## W

Wait (ms) VI, 6-9, 6-10
Wait+(ms) VI
finite pulse train generation, 24-15
measuring frequency and period, 26-4
stopping counter generations, 24-24
Wait on Occurrence function, 7-11 to 7-12
Wait Until Next ms Multiple VI
improving control loop performance, 6-10
multiple-channel single-point analog input, 6-5
software-timed analog I/O control loops, 6-6
waveform acquisition. *See* buffered waveform acquisition.
waveform generation. *See* buffered waveform generation.
Web support from National Instruments, B-1 to B-2
online problem-solving and diagnostic resources, B-1
software-related resources, B-2
Wheatstone bridge, 21-13
Windows environment
installation and configuration
DAQ devices, 2-14
SCXI hardware, 2-18
SCXI software, 2-18
LabVIEW data acquisition hardware support (table), 2-4 to 2-13
NI-DAQ driver, 2-3
Worldwide technical support, B-2
Write N Updates example VI, 11-3
Write to Digital Line VI, 16-2
Write to Digital Port VI
digital output application, 21-19
immediate digital I/O, 16-2
Write to Spreadsheet File VI, 7-8